Escola de Camins
Escola Tècnica Superior d'Enginyeria de Camins, Canals i Ports
UPC BARCELONATECH

TREBALL FINAL DE MÀSTER

Numerical Modelling and Analysis of Pore Architecture in Bone Regeneration Scaffolds

Treball realitzat per:
Pere Calvet Litzell

Dirigit per:
Pedro Díez Mejía
Alberto García González

Màster en:
Enginyeria de Camins, Canals y Ports

Barcelona, 30 de juny de 2020

Departament d'Enginyeria Civil i Ambiental

Numerical Modelling and Analysis of Pore Architecture in Bone Regeneration Scaffolds

By
Pere Calvet-Litzell

A thesis presented for the degree of
Master of Science in Civil Engineering

Escuela de Caminos, Canales y Puertos de Barcelona
Universidad Politécnica de Cataluña
Spain
Academic Year 2019-2020

"Un jour je m'en irai sans en avoir tout dit"

Louis Aragon

Acknowledgements

This document would not be complete without the expression of my most heartfelt gratitude to all the people who have been involved with this thesis. Therefore, I would like to thank Professors Antonio Huerta and Pedro Díez for giving me the opportunity to embark in this great adventure. I cannot forget Professor Maria-Pau Ginebra and Dr. Montserrat Español who have provided such an enthralling topic. Last but not least, I am profoundly indebted to Professor Berto García for his help and commitment. Without his guidance I would not have been able to carry on with the project.

I wish to dedicate my thesis to my family: my parents, Pere and Isabel, and my sister, Maria. Their support during all these years has been most valuable.

Finally, as this project marks the end of a journey that started six years ago, I would like to mention some of the people I have had the opportunity to meet along the way: Erik Bröte, Giovanni Cognetti, Cloe Cortés, Luísa Gil, Paloma Gómez de Olea, Edoardo Marzorati, Roger Miralles, David Nze Ndong and Marcello Vaccarino. They have made it more enjoyable and enriching. They will always have my gratitude and my frienship.

Contents

Abstract

Bone regeneration constitutes a major field within tissue engineering and it is receiving widespread attention. Bone fractures are often difficult to heal. Traditional repair methods usually yield unsatisfactory results. One of the most recent techniques are the so-called scaffolds. These scaffolds are small porous pieces made of Calcium Phosphate. Once inserted in the bone fracture, they foster bone formation by, among other things, releasing Ca^{2+} ions. Pore geometry is believed to play a crucial role by helping to retain the ions in the scaffold. Therefore, determining the optimal pore geometry is key for a standardized production of scaffolds through, for example, 3D printing. Numerical simulation offers the possibility to test different designs at a very low cost. A numerical model that would be able to reproduce the behaviour of the scaffolds would be a very powerful tool. In this thesis, we have put forward a simple model to reproduce the evolution of the concentration. Using the advection-diffusion equation we have modelled the evolution of Ca^{2+} ions in two different scaffolds: a foam scaffold (random structure and difficult to produce)and 3D printed scaffold (ordered structure and easy to produce). The performance of these scaffolds was tested *in vivo* by Barba et al. (see Barba, Maazouz, et al. 2018). The simulations that we have performed seem to match the experimental results: higher concentrations appear in the foam scaffold. Additionally, we have tested the evolution of the concentration under three different velocity fields. For the foam scaffold, the best results are obtained with a simple uniaxial flow, normal to one of the faces of the scaffold. As for the 3D scaffold, the highest concentration is obtained with two orthogonal flows and normal to the faces. The implementation of the equation has been done using FEniCS, an open-source platform to solve PDEs.

Chapter 1

Introduction and Motivation

The damage of tissue and organs is a central problem in human healthcare. Kidney failure, bone fracture or severe burns are just some of the many examples to be found. The traditional solutions to these problems are mainly transplants, surgical reconstruction and mechanical devices. However, they all present shortcomings.

First of all, transplants rely on donors. Figures show a permanent shortage of donors. According to the US Health Resources and Services Administration, in 2019 there were 19 267 donors and 112 568 patients on a waiting list (Division of Transplantation 2020). In Spain, the number of donors was 2 302 in 2019 (Organización Nacional de Transplantes 2019). In 2019, around 59 000 EU citizens were waiting for an organ ([1]). These numbers let us assume that, although the number of donors is increasing, it will certainly be difficult to satisfy the needs of all patients.

Regarding surgical reconstruction, long-term problems appear after such interventions. Among them, serious diseases like cancer (Langer and J. Vacanti 1993).

Finally, mechanical devices such as artificial kidneys cannot fully replace the role of human organs. These devices only fulfil one the many functions an organ performs. Moreover, patients using them cannot have regular lives. For example, in the case of artificial kidneys, they are required to attend dialysis sessions every week in healthcare centres.

A possible solution to these issues could be the production of new human tissue. The idea is to create compatible organs (e.g. kidneys) and tissue (e.g. skin or bone) that are ready for transplant. In the case of tissue, the ultimate goal would be to foster the *in situ* recovery with pre-damage characteristics. Indeed, it is well-known that the natural repair of skin or bone by the body itself (i.e. without any exogenous help) yields imperfect results. Since the 1970's, several researchers have made some advancements in this direction. However, it was not until 1993, with Langer and Vacanti's seminal article (Langer and J. Vacanti 1993) that the field started to receive widespread attention. It was also this article that popularised the term "Tissue Engineering".

In order to present the main aspects of tissue engineering, we shall start by providing an overview on essential biological topics.

1.1 Cells and Cell Differentiation

Cells are considered to be the smallest form of life, they are the basic "building block" of all living things. The human body is itself an assembly of trillions of cells (around thirty trillion). A cell has three main parts: the membrane, the nucleus and the cytoplasm (Figure 1.1). The membrane is the outer layer of the cell. It is responsible for regulating the entrance and exit

[1] Ministerio de Sanidad 2019.

of substances into and out of the cell. The nucleus is the core of the cell and contains genetic information (the cell's DNA). Finally, the cytoplasm is a fluid located between the membrane and the nucleus. It is in the cytoplasm that proteins are produced.

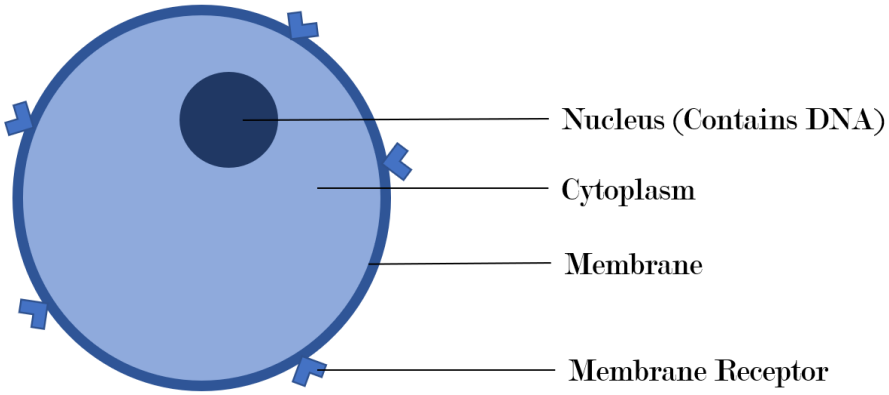

Figure 1.1: Different Parts of a Cell

Cells found in the human body can have different appearances (Figure 1.2). This is due to the fact that they perform different tasks inside our organisms. Just to give an example, erythrocytes (i.e. red blood cells) and neurons are both cells but they have completely different functions.

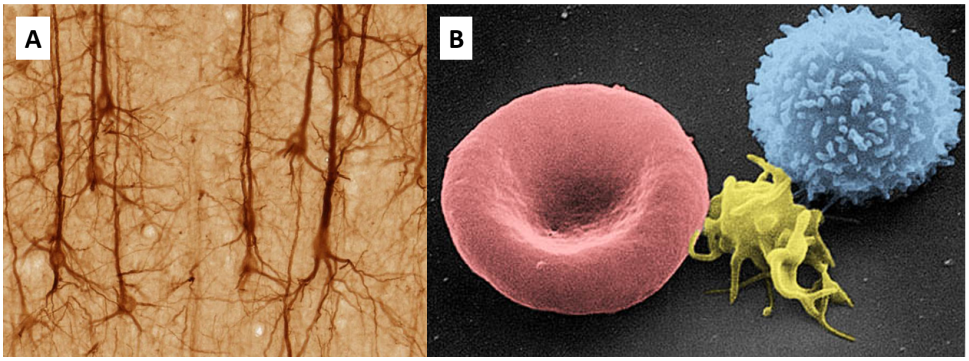

Figure 1.2: Example of two different types of cells. Neurons (A) and a Red Blood Cell (B, in red). Source: UC Regents Davis campus and NCI-Frederick.

Tissue engineering rests on one major biological process called "cell differentiation" (Figure 1.3). It is through this process that the human body produces "specific cells" such as blood, brain, bone or kidney cells. Its starting point is the production of stem cells. These are cells with no specific function which can divide in order to produce new identical cells. Afterwards, a series of mechanisms "differentiate" (i.e. transform) the stem cells into specific cells. The main objective pursued by scientists is to master division and, more importantly, differentiation processes. The understanding of those natural mechanisms is key as it allows to replicate or accelerate them.

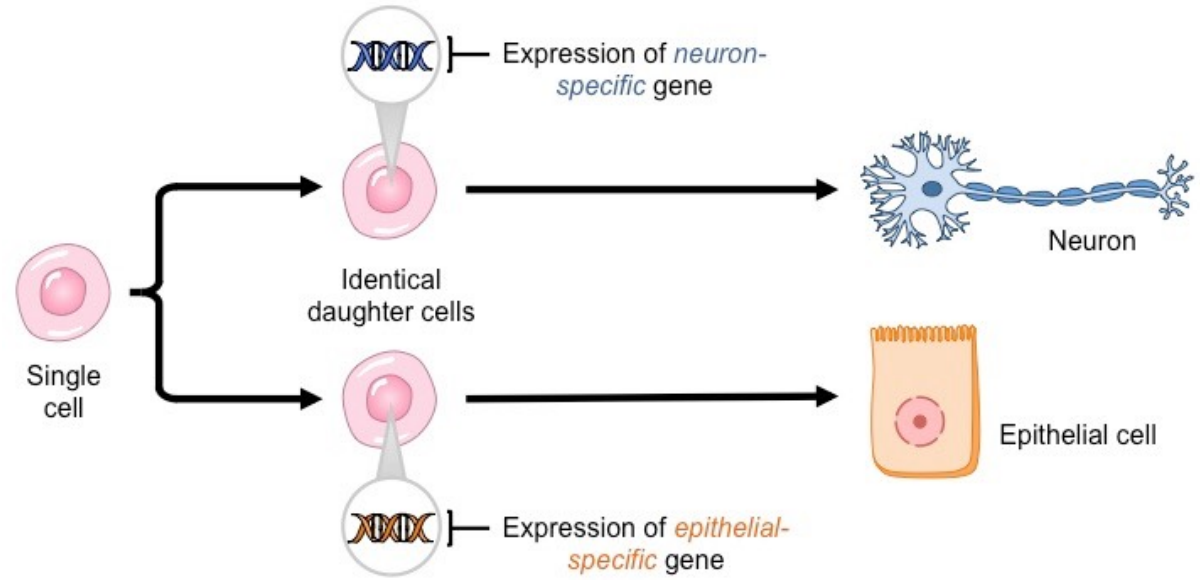

Figure 1.3: Cell Differentiation (Source: http://ib.bioninja.com.au).

There are two main types of stem cells: Adult Stem Cells (ASC) and Embryonic Stem Cells (ESC) (see Evans, Gentleman, and Polak 2006). The main difference is that ESCs are only found in the human embryo. Adult Stem Cells are in turn divided in two types, Hematopoietic Stem Cells (HSC) and Mesenchymal Stem Cells (MSC). HSCs are at the origin of blood cells while MSCs are found in human bone and can become bone, fat and cartilage cells. From a tissue engineering perspective, MSCs present two shortcomings: they only divide a finite number of times and they accumulate genetic changes (cells resulting from division might not be identical).

As for ESCs, they are "more versatile" and can divide an infinite number of times. They can be found in the blastocyst, "a ball of cells formed several days after fertilization [of the egg cell]". However their differentiation has been described as a "hit-and-miss affair", thus stressing the difficulties encountered in mastering the process (Evans, Gentleman, and Polak 2006).

1.2 Tissue Engineering Techniques

As stated before, all tissue engineering techniques are based on cell differentiation. The main difference among these techniques is whether differentiation is *in vivo* or *in vitro*. With the former, differentiation takes place in the patient's body while, with the latter, already differentiated cells are implanted. Both have their advantages and disadvantages. *In vivo* methods use tissue-inducing substances that trigger division and differentiation. If not properly controlled these substances may cause chaotic cell reproduction, eventually resulting in severe diseases such as cancer. On the other hand, *in vitro* methods may face compatibility issues. Indeed, the newly implanted cells may be rejected by the body. Moreover, those cells have shown low efficacy and high mortality rates after implantation.

Most tissue engineering techniques require the use of so-called "scaffolds". Scaffolds are three-dimensional environments for cells to grow and form a functional tissue. They serve as material support on which cells and tissue can rest (Figure 1.4 and Figure 1.5).

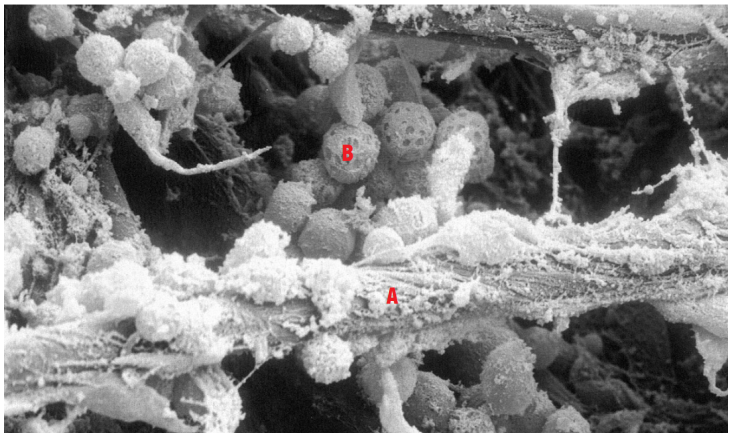

Figure 1.4: Scanning electro micrograph of cells (B) seeded onto synthetic scaffold made of fibres (A).(Source: C. Vacanti 2006).

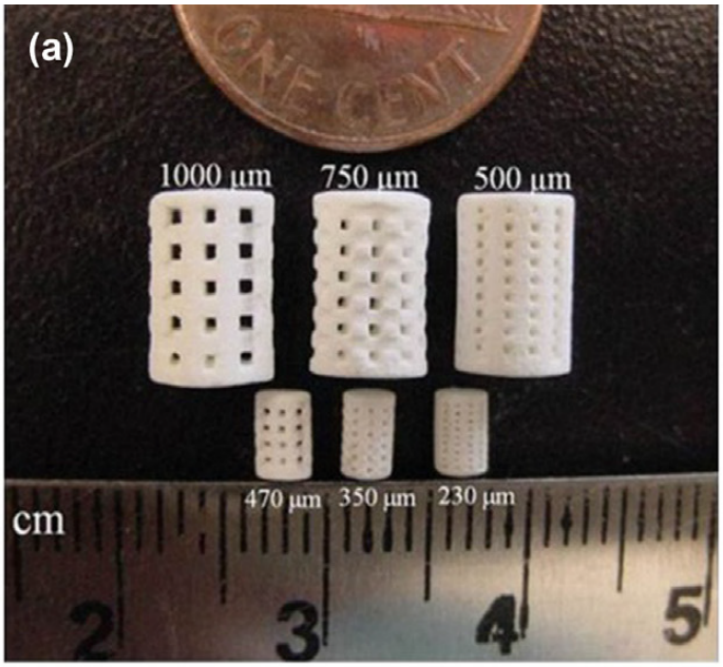

Figure 1.5: Scaffolds intended for bone regeneration (Source: Bose, Vahabzadeh, and Bandyopadhyay 2013).

1.3 Bone Regeneration

One of the most important fields within tissue engineering is bone regeneration. This is due in part to the ageing of the global population. In 2050, 25% of Europeans could be aged 65 or over. With age, the body's capacity to repair bone by itself diminishes dramatically. Indeed, bone fractures are often difficult to heal and traditional repair methods usually yield unsatisfactory results. Hence, the need for improved regeneration techniques.

The specificities of bone regeneration are related to the different cells that constitute bone tissue. There are three types of bone cells: osteoblasts, osteoclasts and osteocytes. Osteoblasts generate new bone whereas osteoclasts are responsible for the re-absorption of old bone tissue. Indeed, bones are constantly undergoing a repair and maintenance process. The process is controlled by osteocytes which sense cracks and pressure on the bone and direct the combined action of osteoclasts and osteoblasts.

As stated before, bone cells are Mesenchymal Stem Cells (MSCs) that have undergone differentiation. This differentiation is triggered by the so-called Bone Morphogenetic Proteins (BMPs) which are naturally present in the human body. BMPs bind to receptors located on the membrane of MSCs (Figure 1.1). Afterwards, receptors send a signal to the nucleus of the MSC. This signal regulates the transcription of certain genes, resulting in differentiation of the MSC.

Research has shown the importance of a particular type of BMP called BMP-2 (Tang et al. 2017). BMP-2s are produced by MSCs and bind to the receptors of these very same cells in what is called the "BMP2 autocrine loop". This loop seems to be triggered by the presence of Ca^{2+} and PO_4^{3-} gradients. As Tang points out, "the osteoinductive [2] Ca-P stimulated BMP-2 autocrine loop in MSCs may be the very factor which initiates the osteoblast differentiation" (see Tang et al. 2017). Thus, the presence of certain ions in the cell's environment triggers its differentiation.

The understanding of these natural mechanisms has enabled the development of tissue engineering techniques. In the case of bone regeneration, Calcium Phosphate (CaP) scaffolds are the most common. They have good osteoinductive properties (i.e. they trigger the differentiation of stem cells into bone cells), thus fostering bone repair. It is important to point out that the most interesting scaffolds are those capable of "material-associated osteoinduction" (see Barba, Diez-Escudero, et al. 2017), which avoids the use of exogenous differentiation factors (e.g. exogenous BMPs) that can have negative collateral effects on the patient. Indeed, these factors can deregulate the cell signalling system and cause uncontrolled cell growth. Moreover, these scaffolds do not serve as a support of external cells. The sole scaffold is implanted in the patient's body. Then, MSCs are expected to gather, divide and differentiate in the scaffold.

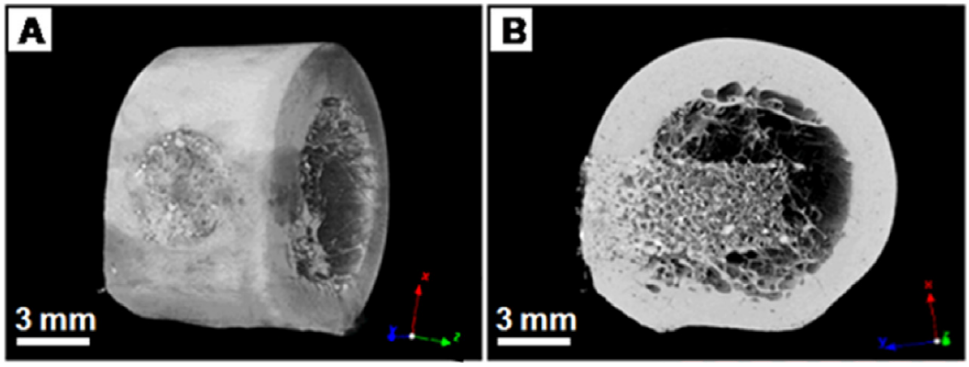

Figure 1.6: Micro-CT 3D images of a CaP scaffold implanted in bone.(Source: Barba, Maazouz, et al. 2018).

Barradas et al. have identified three main characteristics that are responsible for a scaffold's osteoinductive capacity: its microstructure (i.e. pore size and shape), its nanoscale features and its chemical composition (see Barradas et al. 2011). Figure 1.7 summarises the impact of such properties.

The microstructure helps conveying nutrients for the cells whereas the nanoscale features regulate the interaction between the cells and the scaffold. Nanoscale features are also thought to induce differentiation (see Tang et al. 2017). They might adsorb bone-related proteins and allow a selective adsorption of Bone Morphogenetic Proteins (BMPs).

Finally, differentiation is also triggered by chemical messengers released during the scaffold's degradation (the scaffold progressively "dissolves"). Scaffolds release Ca^{2+} and PO_4^{3-} ions. According to Tang, Ca^{2+} gradients are a "potent chemical signal for cell migration and directed growth" (Tang et al. 2017). They act as a "homing signal" that brings together non-differentiated cells for bone remodelling in a specific site. These cells will then proliferate and

[2] Osteoinduction is the process of triggering the differentiation of stem cells into bone cells

differentiate into osteoblasts. PO_4^{3-} ions play a similar role, they also have osteoinductive properties and take part in the mineralization of the bone matrix.

The combined action of the three characteristics is crucial for the formation of an apatite layer on the scaffold (Figure **??**). Apatite is a mineral containing calcium which facilitates the adsorption of BMPs. Hence, its central role in differentiation.

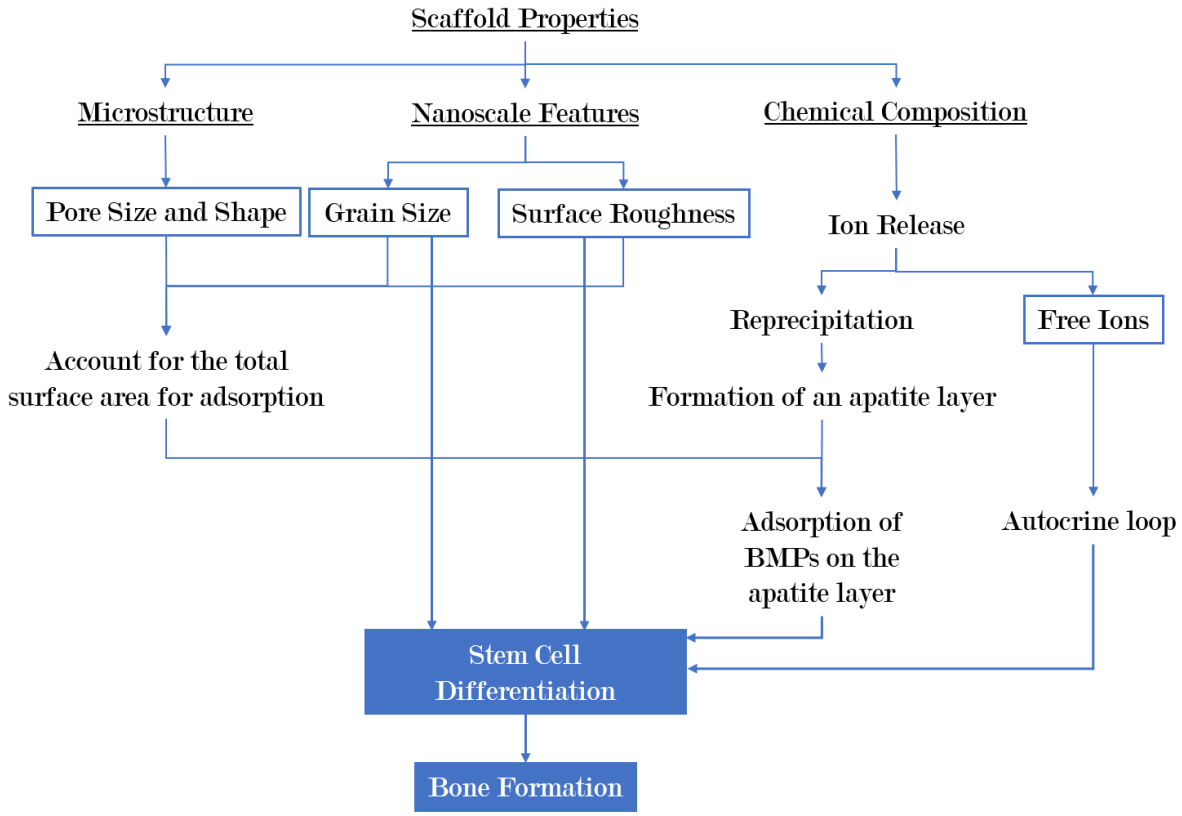

Figure 1.7: Schematic summary of the different scaffold characteristics leading to bone formation.

Present research focuses in finding the optimal scaffold by tuning the three characteristics mentioned above. Barba et al. (see Barba, Diez-Escudero, et al. 2017) tested different materials and pore architectures (i.e. pore size and shape).

With regards to materials, the best results were obtained for calcium-deficient hydroxyapatite (CDHA) which is similar to CaP but has a different nanostructure.

As for pore architecture, Barba et al. used foamed and 3D-printed scaffolds. Foamed scaffolds, as their name suggests, are obtained after the solidification of foams. This results in rather small pores (120 μm) with a great variability in size and shape. Conversely, 3D-printed scaffolds present larger (250 μm) regular pores. The size of these pores is limited by the needle used by the printer, which is too large to produce features below a certain threshold. Foamed scaffolds produced the best results. As a matter of fact, no bone formation was observed in 3D-printed scaffolds. This was confirmed again by Barba et al. (see Barba, Maazouz, et al. 2018) by comparing the different architectures and using only one material (which implies same nanostructure for both scaffolds). The results of their work can be seen in Figure 1.8.

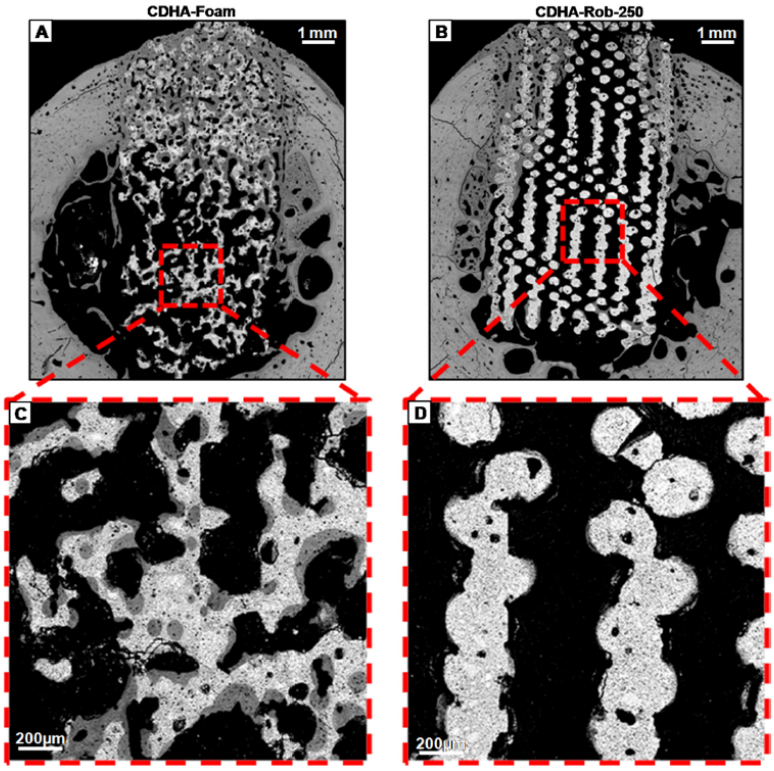

Figure 1.8: Results obtained by Barba, Maazouz, et al. 2018. Note the bone formation (dark grey) in the foam scaffold.

1.4 Numerical Methods and Bioengineering

Over the last decades, improvements in hardware have enabled a proliferation of numerical methods. Researchers can now use models which were deemed too costly a few years ago. Such improvements have had a huge impact in computational mechanics. We are now able to work with more accurate geometries and model more complex behaviours. The modelling of the human body is one of such complex problems. In the early 2000's a series of textbooks on the applications of numerical methods to biomedical engineering were published (see King and Mody 2010; Yamaguchi 2000; Dunn, Constantinides, and Moghe 2005 or Kojić et al. 2008). It was clear that all the knowledge accumulated over the past years in the field of numerical methods could now be used in biomedical engineering. Nowadays, numerical methods have even been used to study the nuclear envelope of cells (see García-González et al. 2018).

One of the most popular methods is the Finite Element Method (FEM). Although it can trace its origins much earlier, the method gained impetus in the 1960s and 1970s with the works of outstanding researchers such as O. C. Zienkiewicz [3]. Since then, the method has gained a central role within numerical simulation. It is used in a wide variety of fields such as the automotive industry, where it is used to simulate crash tests. In the last two decades, it has been introduced in biomedical engineering to study the mechanical properties of bone and soft issues (see Yang 2019). It has also been used to model the behaviour of joints (see Beidokhti et al. 2016). Therefore, it comes as no surprise that the FEM has been used in tissue engineering as well (Boccaccio et al. 2011). For example, back in 2006, Lacroix et al. studied the mechanical resistance of bone scaffolds using the FEM (Lacroix et al. 2006).

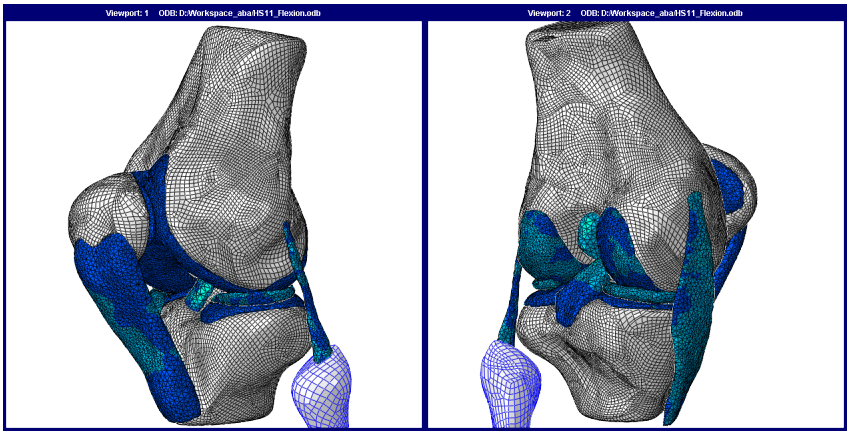

Figure 1.9: Finite Element model of a knee joint (Source: Beidokhti et al. 2016).

1.5 Motivation

Finding the optimal geometry is a necessary step towards mass-production of bone scaffolds. The geometry must have osteoinductive properties but also be easy to produce. More precisely, it should be suited for three-dimensional printing.

Three main motivations lie behind this study. On the one hand, it aims at providing a better understanding of bone scaffolds. The ultimate goal is to identify which characteristics would help to provide better designs. In order to achieve a widespread use of bone scaffolds, these must be effective and easy to produce. This can only be attained through a complete understanding of the underlying mechanisms taking place in scaffolds.

On the other hand, numerical modelling represents a huge reduction in research costs. Research in tissue engineering is has high expenses, specially when it is conducted *in vivo*. Using

[3] Zienkiewicz founded the International Journal for Numerical Methods in Engineering in 1968.

numerical methods to simulate biological processes and test new techniques could help reduce the number of tests, both *in vivo* and *in vitro*.

Finally, it aims at strengthening the ties between computational science and biomedical research. This study has benefited from the collaboration of Dr. Maria-Pau Ginebra and Dr. Montserrat Español from the Department of Material Science at Universtat Politècnica de Catalunya. Drs. Ginebra and Español co-authored the two cited papers by Albert Barba (2017 and 2018). Their work in the field of bone regeneration has been widely acclaimed by several prizes such as the Racquel LeGeros Prize of the International Society for Ceramics in Medicine. In addition, Dr. Ginebra has founded Mimetis Biomaterials, a company specialising in the commercialisation of bone grafts. Working closely with them has provided us with valuable insights and a more precise understanding of the current research in bone regeneration.

1.6 Objectives

With this project we shall try to compare the two different structures used by Barba et al. using numerical methods. More specifically, we will model the evolution of the concentration of Ca^{2+} ions in these two scaffolds. We expect to see higher concentrations in the areas were more bone formation was observed by Barba. The objective is to see if the experimental results can be replicated by model. As a first approach, we present a simple model based on the advection diffusion equation and the potential flow. We expect to be able to retrieve some insights from it. Our hope is that this first small step will develop into a more elaborated model.

Finally, the computational part of the project has been done using FEniCS, an open-source computing platform to solve partial differential equations. FEniCS constitutes an excellent tool and we hope that its use in the scientific community will grow.

This document is organised as follows. Chapter 2 introduces the equations used to model the evolution of the concentration. Chapter 3 presents the Finite Element Method, which has been used to solve them. The results are shown in Chapter 4 and discussed in Chapter 5. The last chapter is a conclusion of the study which also suggests future improvements in the study of bone scaffolds.

Chapter 2

Mathematical Model

2.1 Hypothesis and Problem Statement

We wish to model the evolution of the concentration of calcium ions (Ca^{2+}) in the interstitial fluid that goes through the scaffolds. Higher concentrations are thought to foster cell differentiation through the production and adsorption of BMPs in the scaffold (see Section 1.3). More differentiation among stem cells would in turn lead to a higher production of bone tissue. The geometry of the scaffold is expected to play a key role in the evolution of the concentration.

The evolution of the concentration is studied as a classical advection-diffusion problem. This implies that there are two mechanisms. On the one hand, the scaffold releases ions that start to fill the interstitial fluid that goes through it. On the other hand, the fluid drags the ions, thus changing the concentration. To model the fluid's velocity a second assumption has been made. As a first approach, we have chosen a potential flow for its simplicity.

The solution of the advection-diffusion equation yields the concentration of calcium ions in a given volume of interstitial fluid.

The equation will be solved for two scaffolds (i.e. two domains). It is important to recall that the domain is not the scaffold itself but the bodily fluid inside it. Therefore, figures shown throughout this chapter depict volumes of fluid. In addition, for each scaffold a steady state and a transient analysis will be performed. The first scaffold is made of a solidified CDHA[1] foam. This implies that pores are irregular and have a diameter close to 120 μm. Although using the same material, the second one is 3D printed. Its pores are regular with a diameter of 340 μm and separated by 250 μm.

As stated before, this study has been motivated by Barba's work (see Barba, Maazouz, et al. 2018). Barba found that bone formation only took place in the "foam scaffold". Therefore, simulations are expected to show higher concentrations in that scaffold. Indeed, higher concentrations would trigger more cell differentiation which in turn would lead to more bone production.

Finally, we wish to point out that the degradation of the scaffold is not modelled.

[1] Calcium-Deficient Hydroxyapatite

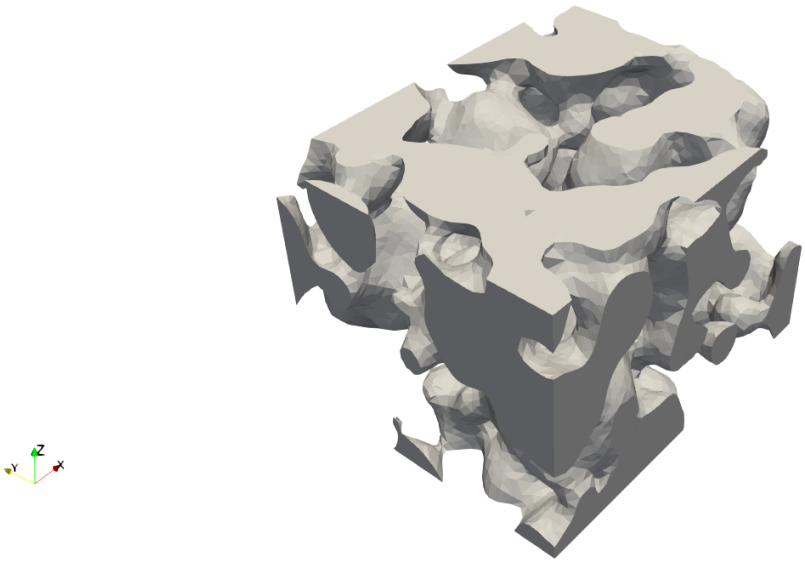

Figure 2.1: Volume of interstitial fluid inside a foam scaffold (STL digital model obtained through Micro-CT scanning).

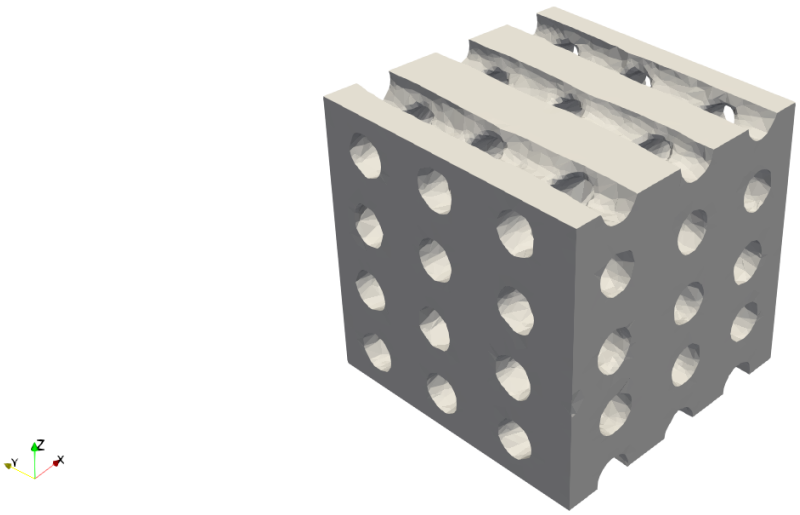

Figure 2.2: Volume of interstitial fluid inside a structured scaffold (STL digital model obtained through Micro-CT scanning).

2.2 Advection-Diffusion Equation

The general advection-diffusion equation for mass transfer can be written as follows:

$$\underbrace{\frac{\partial u}{\partial t}}_{Concentration's Evolution} = \overbrace{\nabla \cdot (\mu \nabla u)}^{Diffusion} - \underbrace{\nabla \cdot (\mathbf{v}u)}_{Advection} + \overbrace{s}^{SourceTerm} \tag{2.1}$$

where

- u is the concentration of the chemical species (i.e. ions),
- μ is the diffusivity coefficient usually expressed in m^2/s,
- $\mathbf{v}$ is the velocity field of the fluid,
- s is the so-called source (or sink) term, which accounts for the creation (if $s > 0$) or destruction (if $s < 0$) of the chemical species.

To put it in a simple manner, the equation states that the evolution of the concentration depends on three phenomena: diffusion, advection and chemical reactions.

Diffusion is caused by a gradient of concentration. Chemical species "move" from the parts of the fluid with higher concentration to the parts with lower concentration. This can be seen when a drop of a coloured substance falls in a glass of water. Although water is still, the coloured substance expands.

Advection is the transport of mass by a fluid. The chemical species of interest (whose concentration is u) moves with the fluid. This can be seen in the advection term which includes the velocity field $\mathbf{v}$.

Finally, chemical reactions can lead to the formation or destruction of the species of interest. These are caused by the interactions with other species. This last phenomenon shall not be considered in our model as it does not take into account the interactions of calcium ions with other species. Therefore $s = 0$.

2.2.1 Velocity Field

As stated above, the velocity field is considered to derive from a velocity potential ϕ. This means that it is equal to the gradient of the scalar function ϕ (see equation 2.2). In addition, the fluid is also considered to be incompressible which implies that the divergence of $\mathbf{v}$ is naught (see eq. 2.3).

Therefore, using mathematical notation, we have

$$\mathbf{v} = \nabla \phi \tag{2.2}$$

and

$$\nabla \cdot \mathbf{v} = 0. \tag{2.3}$$

These two equations can be rewritten as

$$\nabla \cdot (\nabla \phi) \implies \Delta \phi = 0. \tag{2.4}$$

In addition, the boundary conditions of the problem are:

$$\phi(x, y, z) = x \ for \ \Gamma_1 \cup \Gamma_2 \ , \tag{2.5}$$

where Γ_1 and Γ_2 are the two faces whose normals are parallel to the x-axis. Such boundary conditions ensure that the fluid flows along the X-axis and that vectors of the velocity field will

have a norm close to the unit. It is then possible to multiply the field by a constant to obtain the desired velocity. For example, the velocity of the fluid is around 1000 $\mu m/s$ (see Garcia and Ducheyne 1994). In order to simulate the evolution of the concentration under different circumstances, other boundary conditions have been used, yielding different velocity fields (see Appendix).

Figures 2.3, 2.4 and 2.5 illustrate the idea of the potential flow on a simple square domain with three holes. Figure 2.3 is the potential flow ϕ obtained by solving Equation (2.4) with Boundary Conditions (2.5). Figure 2.4 shows the velocity field which is equal to the gradient of ϕ. Finally, Figure 2.5 depicts the streamlines of the potential flow.

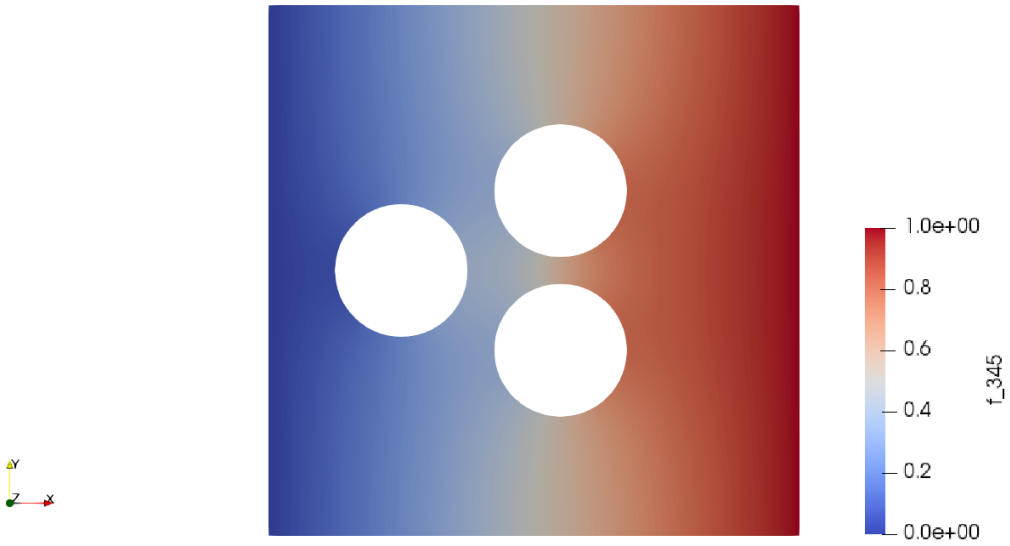

Figure 2.3: Velocity Potential ϕ resulting from solving Equation (2.4).

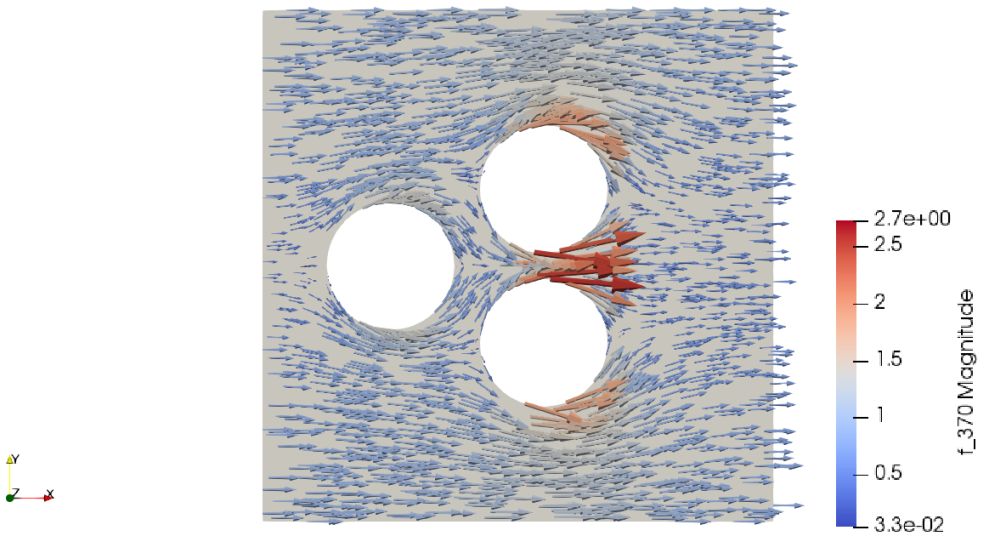

Figure 2.4: Velocty Field $\mathbf{v}$ equal to the gradient of ϕ.

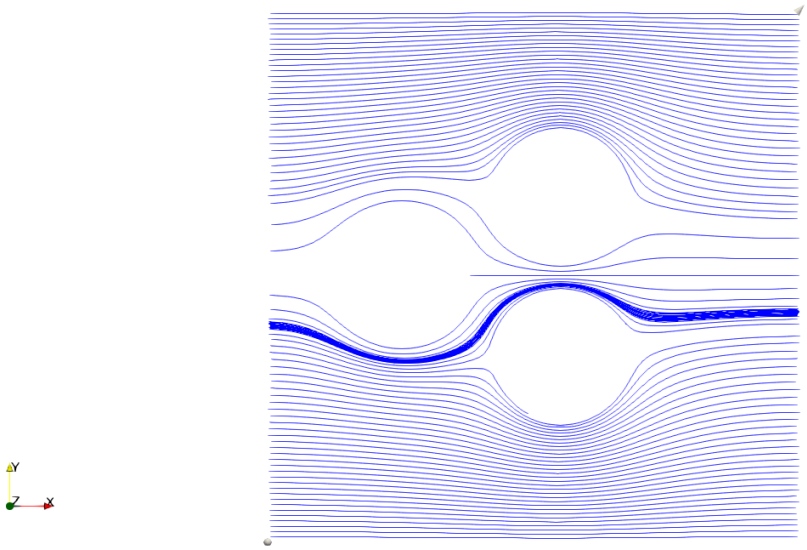

Figure 2.5: Streamlines of the potential flow.

2.2.2 Diffusion coefficient

Diffusion depends on the so-called diffusion coefficient μ, which takes a different value depending on the chemical species. It can be obtained using the Stokes-Einstein equation:

$$\mu = \frac{k_B T}{6\pi\eta r} \ , \tag{2.6}$$

where

- k_B is the Boltzmann constant $k_B = 1.38 \times 10^{-23}\,\mathrm{J\,K^{-1}}$,
- T is the absolute temperature,
- η is the dynamic viscosity,
- r is the radius of the spherical particle.

The body's temperature is 37 °C which in Kelvin is 310 K. The dynamic viscosity of the interstitial fluid is $1.5 \times 10^{-3}\,\mathrm{Pa\,s^{-1}}$ and the ionic radius of Calcium (as the scaffold releases Ca^{2+} ions) is $r = 1.14 \times 10^{-10}\,\mathrm{m}$ (for these values see Garcia and Ducheyne 1994). The values expressed in the International System of Units yield $\mu = 1.33 \times 10^{-9}\,\mathrm{m^2\,s^{-1}}$.

As the units should be in accordance with the scale of the geometry used in the simulation, the units of μ have to be changed. From the digitalization process of the real scaffolds, it is known that one unit in the STL (i.e. the file resulting from the scan) equals u=10 μm. First we convert μ to square micrometers per second.

$$\mu = \frac{1.33 \times 10^{-9}\ m^2}{1\ s} \cdot \frac{1 \times 10^{12}\ \mu m^2}{1\ m^2} = 1330\ \mu m^2/s \tag{2.7}$$

However the value has to be divided by 10^2 as one unit equals 10 μm. Finally $\mu = 13.30\ u^2/s$.

2.3 Boundary Conditions

The studied domain is a volume of fluid going through the scaffold. The release of ions by the scaffold is modelled by a Dirichlet boundary condition: the boundaries of the volume that are in contact with the scaffold have constant and maximum concentration (i.e $u = 1$). This means that the scaffold is constantly releasing ions. In addition, a Dirichlet boundary condition such as $u = 0$ (i.e. concentration is zero) is imposed on the inflow faces of the volume. This is done to avoid diffusion towards that area. Thus boundary conditions are:

- $u = 1$ on Γ_w
- $u = 0$ on Γ_i

where Γ_w are the boundaries in contact with the scaffold (Figure 2.6 in red) and Γ_i are the inflow faces (the face in the YZ-plane, on the left in Figure 2.6).

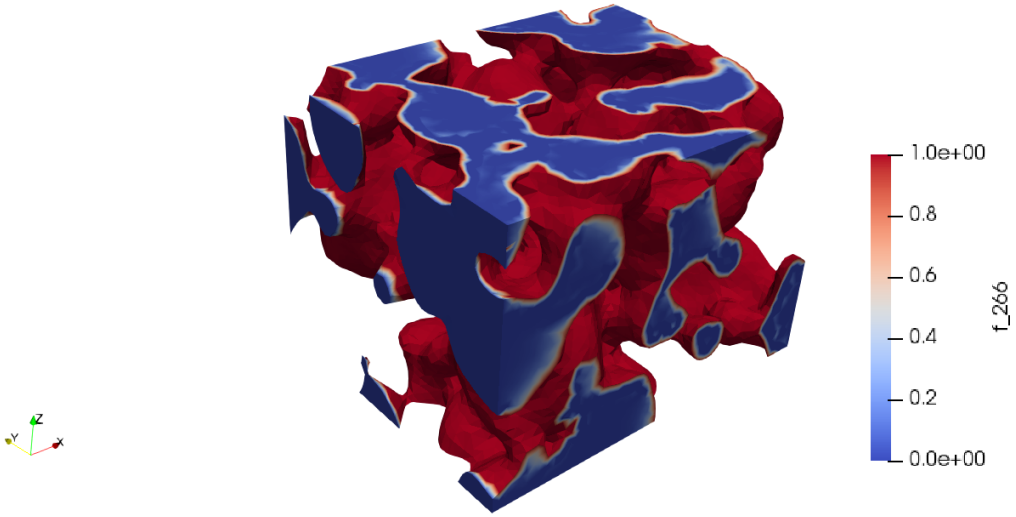

Figure 2.6: Boundary conditions of the problem.

Chapter 3

Finite Element Method

In the previous chapter the PDE used to model the evolution of the concentration was presented. It was also pointed out that another PDE had to be solved in order to obtain the velocity field that was used in the main equation. Both equations can be solved using the Finite Element Method (FEM), which is a widespread method in computational engineering. An overview of the method is given in the following pages. Stabilization and transient state problems are also discussed. Readers already familiar with the FEM can go directly to the last section of the chapter where the implementation is discussed.

Note: As the advection-diffusion is widely used equation, different readers may be familiar with different notations. In the following sections we shall use "$\boldsymbol{v}$" for the velocity field and "u" and "w" for the trial and test funcions, respectively.

3.1 FEM Basics

The main idea behind the Finite Element Method is discretization. Since the PDE has not an analytical solution, we shall try to approximate its values in certain points of the domain. The solution over the whole domain can then be obtained through interpolation. In mathematical terms, the FEM "rests upon the discrete representation of a weak integral form of the partial differential equation to be solved" (Donea and Huerta 2003, p.19). Therefore two steps are required: formulate the weak form and discretise it.

3.1.1 Weak Form

The weak form of the problem is obtained by multiplying the PDE by a test function w and integrating over Ω. The test function w sould be such that $w \in \mathcal{V}$ where

$$\mathcal{V} = \left\{ w \in \mathcal{H}^1(\Omega) | w = 0 \text{ on } \Gamma_D \right\}. \tag{3.1}$$

In other words, the test function must vanish on the Dirichlet boundary. $\mathcal{H}^1$ is the Sobolev space of functions which are square integrable and have first derivatives which are also integrable over Ω [1].

Velocity Field

The weak form of equation 2.4 is obtained through integration by parts:

$$\int_\Omega \Delta\phi w \, d\Omega = \int_\Gamma \frac{\partial \phi}{\partial n} w \, ds - \int_\Omega \nabla\phi \nabla w \, d\Omega \tag{3.2}$$

[1] For a succinct note on Sobolev spaces see Appendix A. For a more detailed explanation see Donea and Huerta 2003.

However as w is zero on the boundary the final variational form is :

$$-\int_{\Omega} \nabla\phi\nabla w \; d\Omega = 0 \tag{3.3}$$

Advection-Diffusion

Let us formulate the weak form of a particular case of the advection-diffusion equation. The situation is considered stationary (i.e. $\frac{\partial u}{\partial t} = 0$) and no source term is taken into account (i.e. s = 0). The diffusion coefficient μ is constant. Therefore the original equation

$$\frac{\partial u}{\partial t} = \nabla \cdot (\mu \nabla u) - \nabla \cdot (\mathbf{v}u) + s \tag{3.4}$$

becomes

$$\nabla \cdot (\mathbf{v}u) - \nabla \cdot (\mu \nabla u) = 0. \tag{3.5}$$

In addition, we assume that there are Dirichlet boundary conditions,

$$u = u_D \; on \; \Gamma_D, \tag{3.6}$$

and that Neumann conditions are natural,

$$\mathbf{n} \cdot \mu \nabla u = 0 \; on \; \Gamma_N. \tag{3.7}$$

We recall that $\partial\Omega = \Gamma_D \cup \Gamma_N$. By multiplying by w and integrating, the following expression is reached:

$$\int_{\Omega} w(\mathbf{v} \cdot \nabla u) \; d\Omega - \int_{\Omega} w\nabla \cdot (\mu \nabla u) \; d\Omega = 0 \tag{3.8}$$

Using integration by parts and the divergence theorem, it is possible to write

$$\int_{\Omega} w\nabla \cdot (\mu \nabla u) \; d\Omega = -\int_{\Omega} \nabla w \cdot (\mu \nabla u) \; d\Omega + \int_{\partial\Omega} w(\mu \nabla u) \cdot \mathbf{n} \; d\Gamma. \tag{3.9}$$

Bearing in mind that $w = 0$ on Γ_D and that Neumann conditions are natural, the expression comes down to

$$\int_{\Omega} w(\mathbf{v} \cdot \nabla u) \; d\Omega + \int_{\Omega} \nabla w \cdot (\mu \nabla u) \; d\Omega = 0. \tag{3.10}$$

Note that the regularity requirements are fulfilled if u and w are trial and test functions respectively. Once the weak form has been obtained, it is possible to perform the spatial discretization using the Galerkin method. Before going any further, a compact form of the weak formulation is presented below:

$$\begin{aligned} a(w, u) &= \int_{\Omega} \nabla w \cdot (\mu \nabla u) d\Omega \\ c(\mathbf{v}; w, u) &= \int_{\Omega} w(\mathbf{v} \cdot \nabla u) \; d\Omega. \end{aligned} \tag{3.11}$$

This yields:

$$a(w, u) + c(\mathbf{v}; w, u) = 0. \tag{3.12}$$

3.1.2 Galerkin Method

In the Galerkin method, trial and test functions are chosen in finite dimensional spaces which are subsets of $\mathcal{S}$ and $\mathcal{V}$. We recall that

$$\mathcal{V} = \left\{ w \in \mathcal{H}^1(\Omega) | w = 0 \ on \ \Gamma_D \right\} \tag{3.13}$$

and

$$\mathcal{S} = \left\{ u \in \mathcal{H}^1(\Omega) | u = u_D \ on \ \Gamma_D \right\} \equiv \mathcal{V} + \{\bar{u}_D\}. \tag{3.14}$$

In mathematical notation, those subsets are $\mathcal{S}^h \subset \mathcal{S}$ and $\mathcal{V}^h \subset \mathcal{V}$. As for the functions, $u^h \in \mathcal{S}^h$ and $w^h \in \mathcal{V}^h$. The test functions w^h must also vanish on the Dirichlet bounadry and the trial functions u^h must verify the boundary conditions. The weak form of the problem becomes:

Find $u^h \in \mathcal{S}^h$ such that

$$a(w^h, u^h) + c(\mathbf{v}; w^h, u^h) = 0, \ \forall w^h \in \mathcal{V}^h. \tag{3.15}$$

In the finite element space, the approximation u^h can be written as

$$u^h(\mathbf{x}) = \sum_{A \in \eta \backslash \eta_D} N_A(\mathbf{x}) u_A + \sum_{A \in \eta_D} N_A(\mathbf{x}) u_D(\mathbf{x}_A) \ , \tag{3.16}$$

where N_A is the shape function associated with node A and u_A is the value of u in that node. u^h can be seen as a projection of u on a finite dimension space defined by the shape functions. On the other hand, test functions w^h are closely related to shape functions since:

$$w^h \in \mathcal{V}^h := \ span\{N_A\}, \ where \ A \in \eta \backslash \eta_D \tag{3.17}$$

The discrete weak form is finally obtained:

$$\begin{aligned} \sum_{B \in \eta \backslash \eta_D} [a(N_A, N_B) + c(\mathbf{v}; N_A, N_B)] u_B = \\ - \sum_{B \in \eta_D} [a(N_A, N_B) + c(\mathbf{v}; N_A, N_B)] u_D(\mathbf{x}_B), \ for \ all \ A \in \eta \backslash \eta_D \end{aligned} \tag{3.18}$$

After assembly operations,this discrete form can be rewritten as an algebraic system:

$$(\mathbf{C} + \mathbf{K})\mathbf{u} = \mathbf{f} \tag{3.19}$$

Where u is the vector of the unknowns. $\mathbf{C}$ and $\mathbf{K}$ are the convection matrix and the diffusion matrix, respectively. They are derived from the convection and diffusion terms of the weak form.

$$\begin{aligned} \mathbf{C} = \mathsf{A}^{\mathrm{e}} \, C^e \quad C^e_{ab} = \int_{\Omega^e} N_a (\mathbf{v} \cdot \nabla N_b) d\Omega \\ \mathbf{K} = \mathsf{A}^{\mathrm{e}} \, K^e \quad K^e_{ab} = \int_{\Omega^e} \nabla N_a \cdot \mu \nabla N_b d\Omega \end{aligned} \tag{3.20}$$

On the other hand, $\mathbf{f}$ results from the Dirichlet boundary conditions.

$$\begin{aligned} \mathbf{f} = \mathsf{A}^{\mathrm{e}} \, f^e \\ f^e_a = - \sum_{b=1}^{n_{en}} [a(N_a, N_b)_{\Omega^e} + c(\mathbf{v}; N_a, N_b)_{\Omega^e}] u_D(x^e_b) \end{aligned} \tag{3.21}$$

Broadly speaking, Galerkin discretization yields a system whose unknowns are the values of u^h at certain nodes of the domain Ω. The values of u^h at other points of the domain are given

by interpolation which is at the base of the construction of u^h (see 3.16). Therefore, solving the system is the ultimate goal of the FEM. Indeed, the system yields the approximation of u. The closeness between u^h and u will be given by the element size h. This is shown through Céa's lemma.

3.2 Transient State Analysis

This section explains how to perform a transient state analysis. The interest of performing a transient analysis is twofold. Firstly, it can give an order of magnitude of the time-scale. In order words, how long does it take to reach the steady state? Secondly, it can show how the ions spread. This would allow us to identify trends and patterns.

In the transient state, the derivatives with respect to time are no longer zero. Therefore, the equation is

$$\frac{\partial u}{\partial t} = \nabla \cdot (\mu \nabla u) - \mathbf{v} \cdot (\nabla u). \tag{3.22}$$

Passing all the terms to the left side it becomes

$$\frac{\partial u}{\partial t} - \nabla \cdot (\mu \nabla u) + \mathbf{v} \cdot (\nabla u) = 0. \tag{3.23}$$

The weak form is then (the velocity field has zero divergence)

$$\int_\Omega \frac{\partial u}{\partial t} w \; d\Omega + \int_\Omega \mu \nabla u \nabla w \; d\Omega + \int_\Omega w(\mathbf{v} \cdot \nabla u) \; d\Omega = 0. \tag{3.24}$$

3.2.1 Time Discretization

In order to solve the transient state, the time derivative is discretized. This means that there will be a finite number of time steps and that for each step a stationary problem will be solved. A way to discretize the derivative is to use the Backward-Euler scheme:

$$\left(\frac{\partial u}{\partial t} \right)^{n+1} \approx \frac{u^{n+1} - u^n}{\Delta t} \tag{3.25}$$

In this case, for a given step, the solution of the previous step is used to calculate the derivative. Therefore, given the value of u^0 it is possible to calculate the following values of u. Using the Backward-Euler scheme, the weak form of 3.22 is:

$$\int_\Omega \frac{u^{n+1} - u^n}{\Delta t} w \; d\Omega + \int_\Omega \mu \nabla u^{n+1} \nabla w \; d\Omega + \int_\Omega w(\mathbf{v} \cdot \nabla u^{n+1}) \; d\Omega = 0 \tag{3.26}$$

Reordering the terms we obtain:

$$\frac{1}{\Delta t} \int_\Omega w u^{n+1} \; d\Omega + \int_\Omega \mu \nabla u^{n+1} \nabla w \; d\Omega + \int_\Omega w(\mathbf{v} \cdot \nabla u^{n+1}) \; d\Omega = \frac{1}{\Delta t} \int_\Omega w u^n \; d\Omega \tag{3.27}$$

The term on the right hand side contains the solution of the previous step u^n. It can be treated as a constant source term. The first term of the left hand side introduces a new matrix in the algebraic system, the mass matrix. Therefore we have:

$$(\frac{1}{\Delta t}\mathbf{M} + \mathbf{C} + \mathbf{K})\mathbf{u}^{n+1} = \mathbf{f} \tag{3.28}$$

The mass matrix is computed as follows:

$$\mathbf{M} = \mathsf{A}^{\mathrm{e}}\, M^e \quad M^e_{ab} = \int_{\Omega^e} N_a N_b \; d\Omega \tag{3.29}$$

Note that now vector $\mathbf{f}$ contains the term associated with u^n. Therefore:

$$\begin{gathered} \mathbf{f} = \mathsf{A}^{\mathrm{e}}\, f^e \\ f^e_a = \frac{1}{\Delta t}(N_a, u^n)_{\Omega_e} - \sum_{b=1}^{n_{en}} [a(N_a, N_b)_{\Omega^e} + c(\mathbf{v}; N_a, N_b)_{\Omega^e}] u_D(x^e_b) \end{gathered} \tag{3.30}$$

3.3 Stabilization

3.3.1 General Aspects

In an advection-diffusion problem, the relative importance of advective and diffusive effects is given by the Péclet number:

$$Pe = \frac{vh}{2\mu} \tag{3.31}$$

Where v is the velocity, μ is the diffusivity and h is the element size. Therefore for a fixed h (i.e. a given mesh), a higher Péclet implies a more advection-dominated problem. For $Pe > 1$ (i.e. advection-dominated problem), oscillations can be observed in the numerical solution when using the standard Galerkin formulation. Stabilization techniques such as the Stream-Upwind Petrov-Galerkin (SUPG) method are then required.

The SUPG method provides consistent stabilization which means that the solution of the PDE is equal to that of the weak form. The main principle of the method is to add an extra term to the weak form. This new term is a function of the residual in order to ensure consistency. Given the general advection-diffusion equation

$$v \cdot \nabla u - \nabla \cdot (\mu \nabla u) + \sigma u = s \; in \; \Omega \; , \tag{3.32}$$

the residual is defined as follows:

$$\mathcal{R}(u) = v \cdot \nabla u - \nabla \cdot (\mu \nabla u) + \sigma u - s = \mathcal{L}(u) - s \; , \tag{3.33}$$

where $\mathcal{L}(u)$ is the differential operator associated with the differential equation. In finite dimensional spaces, $\mathcal{L}(u)$ is only computed for each element interior. Stabilization techniques have the following general form:

$$\begin{aligned} a(w,u) + c(\mathbf{v}; w, u) + (w, \sigma u) + \sum_e \int_{\Omega^e} \mathcal{P}(w) \tau \mathcal{R}(u) d\Omega \\ = (w, s) + (w, h)_{\Gamma_N} \; , \end{aligned} \tag{3.34}$$

where $\mathcal{P}(w)$ is an operator applied to the test function and τ is the so-called stabilization parameter (or instrinsic time). Note the presence of the residual as well. For every stabilization technique $\mathcal{P}(w)$ has a different expression. For SUPG we have:

$$\mathcal{P}(w) = \mathbf{v} \cdot \nabla w \; . \tag{3.35}$$

The stabilization parameter τ can be defined as:

$$\tau = \bar{\mu} / \|\mathbf{v}\|^2 \; , \tag{3.36}$$

where $\bar{\mu} = \beta a h/2$ and $\beta = coth Pe - 1/Pe$. In 2000, Codina (see Donea and Huerta 2003, p.65) proposed a different definition of τ:

$$\tau = \Big(\frac{2a}{h} + \frac{4\mu}{h^2} + \sigma\Big)^{-1} = \frac{h}{2a}\Big(1 + \frac{1}{Pe} + \frac{h}{2a}\sigma\Big)^{-1} \; . \tag{3.37}$$

3.3.2 Particular Case

Having introduced the general aspects of SUPG, it is now possible to focus on a particular case. Recalling that in our problem $s = 0$ and $\sigma = 0$ and that we are interested in the transient state, the problem can be simplified as:

$$\frac{\partial u}{\partial t} - \nabla \cdot (\mu \nabla u) + \nabla \cdot (\mathbf{v} u) = 0 \ , \tag{3.38}$$

being the weak form:

$$\int_\Omega \frac{\partial u}{\partial t} w \ d\Omega + \int_\Omega \mu \nabla u \cdot \nabla w \ d\Omega + \int_\Omega w(\mathbf{v} \cdot \nabla u) \ d\Omega = 0. \tag{3.39}$$

Adding the stabilization term it becomes:

$$\begin{aligned} \int_\Omega \frac{\partial u}{\partial t} w \ d\Omega + \int_\Omega \mu \nabla u \cdot \nabla w \ d\Omega + \int_\Omega w(\mathbf{v} \cdot \nabla u) \ d\Omega \\ + \sum_e \int_{\Omega^e} \mathcal{P}(w) \tau \mathcal{R}(u) d\Omega = 0. \end{aligned} \tag{3.40}$$

The time derivative is approximated by the Backward-Euler scheme which yields:

$$\begin{aligned} \int_\Omega \frac{u^{n+1} - u^n}{\Delta t} w \ d\Omega + \int_\Omega \mu \nabla u^{n+1} \nabla w \ d\Omega + \int_\Omega w(\mathbf{v} \cdot \nabla u^{n+1}) \ d\Omega \\ + \sum_e \int_{\Omega^e} \mathcal{P}(w) \tau \mathcal{R}(u^{n+1}) d\Omega = 0 \ . \end{aligned} \tag{3.41}$$

This weak form provides the system that will be solved at each time step (see Section 3.2). Note that for a time step $n + 1$ the previous solution n is required.

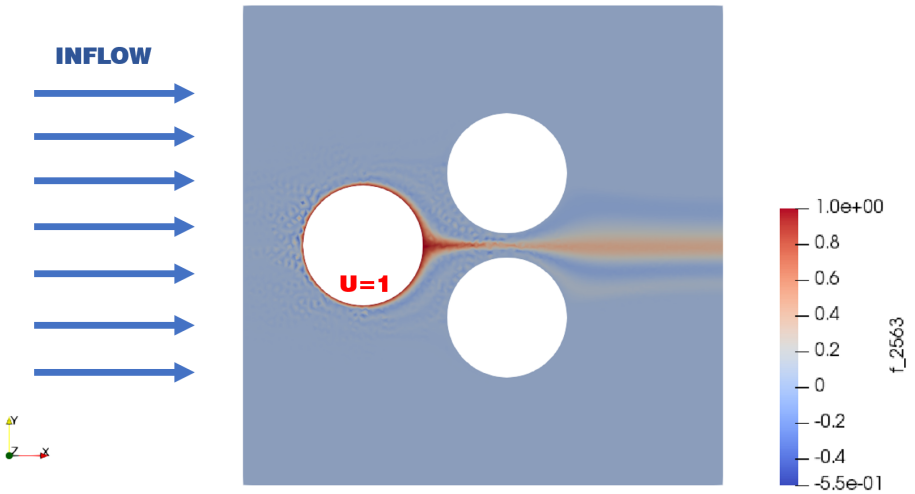

Figure 3.1: Solution of the advection-diffusion equation over a simple domain without using SUPG.

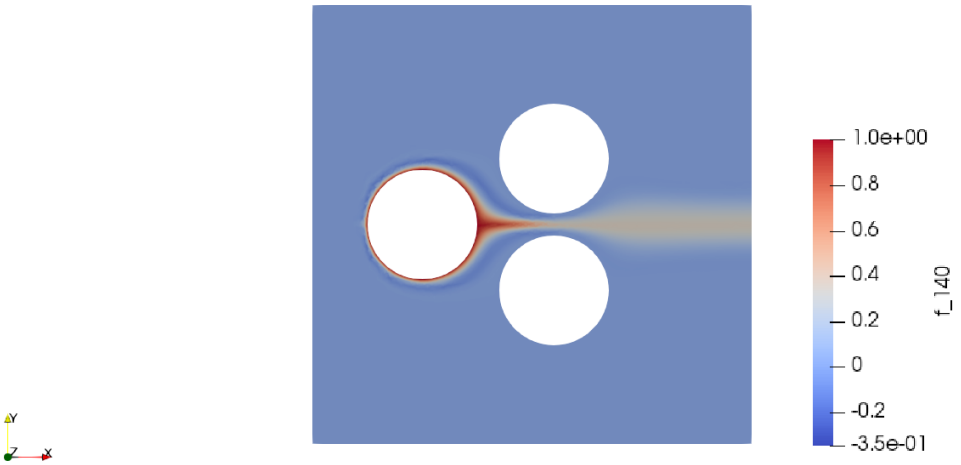

Figure 3.2: Solution of the advection-diffusion equation over a simple domain using SUPG.

3.4 Stopping Criteria

When programming, the time-stepping can be implemented through a logical loop. The most simple way is to establish a fixed number of steps and to solve the problem for every one of them. Needless to say that the number of steps will have an impact on the computational cost of the program. The time increment Δt is the result of the division of the total physical time T (i.e. the time that the advection-diffusion phenomenon is being observed) by the number of steps. Both T and the number of steps are set beforehand.

While this loop allows to have a good control over the number of steps, and therefore the computational cost, it does not ensure that the stationary situation will be reached. An alternative structure, based on the difference between solutions of two successive steps, can lead to the stationary state. The simulation will stop when the difference goes below a desired threshold. In this case two parameters will have an influence on the computational cost: the time increment Δt and the tolerance (the threshold).

For this structure, the time increment has to be set as there is not a fixed number of steps and T is not known in advance (i.e. we do not know how long it takes to reach the stationary state). Bigger time increments will diminish the final number of steps (and thus the computational cost) but information will be lost regarding the evolution between consecutive steps. On the other hand, a higher tolerance will make the program stop earlier. However, the process may not have reached the stationary state.

3.5 Implementation

The PDEs have been solved using FEniCS, an open-source computing platform based on the Finite Element Method. It is distributed through two Python libraries called “fenics” and “dolfin”. FEniCS codes can be written with the help of any Python editor such as Spyder (Figure 3.3).

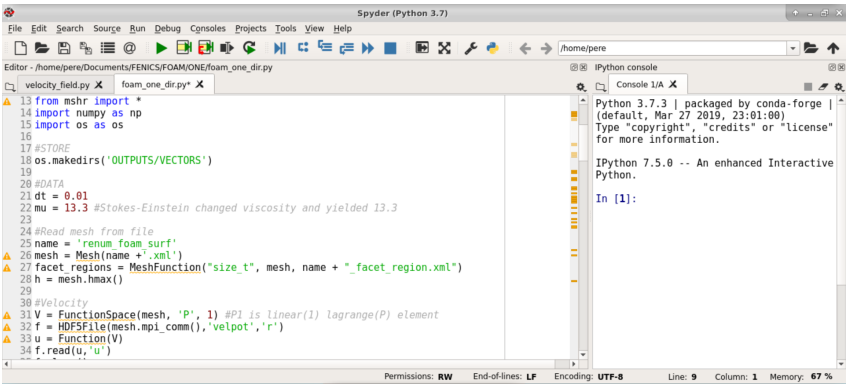

Figure 3.3: Writing a FEniCS code with Spyder. Spyder also allows to run the code.

3.5.1 Obtaining a Tetrahedral Mesh

FEniCS uses the XML format for its meshes. However, the original file is an STL, a very popular format for 3D printing. The STL files used in this study were the result of a micro-CT scan of the scaffolds. Obtaining a proper mesh from an STL file is a cumbersome task. As shown in Figure 3.4, the STL geometry is made of independent triangles and no connectivity is available. Moreover, the resulting figure is hollow. Therefore, the format does not prevent triangles from overlapping or having very close nodes that could be merged. Such flaws are rather common and they must repaired before converting the STL file into a 3D mesh. There are several STL editors, Meshlab and Meshmixer are two of the best and have the advantage of being free to download. For example, Meshlab has a wide variety of tools that allow to close holes in the STL, merge nodes or remove intersecting faces. For Abaqus users, it could be interesting to know that an Abaqus tool called "collapse edge" is also useful to correct the STL's imperfections. It must be pointed out that Abaqus only reads the "ASCII STL" format. Therefore, it is important to make sure that files are exported in this format and not "Binary STL".

```
solid STL generated by MeshLab
  facet normal  7.014307e-01 -4.273050e-01  5.704432e-01
    outer loop
      vertex  -3.693585e+02 -4.185030e+02  3.922917e+02
      vertex  -3.679826e+02 -4.162529e+02  3.922854e+02
      vertex  -3.699898e+02 -4.166705e+02  3.944406e+02
    endloop
  endfacet
  facet normal  3.628388e-01  3.107751e-01 -8.785027e-01
    outer loop
      vertex  -4.815820e+02 -3.895182e+02  4.975193e+02
      vertex  -4.833769e+02 -3.863066e+02  4.979141e+02
      vertex  -4.806494e+02 -3.868084e+02  4.988631e+02
    endloop
  endfacet
  facet normal -9.843393e-02 -9.695827e-01  2.240984e-01
    outer loop
      vertex  -4.026776e+02 -5.317764e+02  5.098958e+02
      vertex  -4.060580e+02 -5.314753e+02  5.097133e+02
      vertex  -4.040664e+02 -5.322538e+02  5.072199e+02
    endloop
  endfacet
```

Figure 3.4: Sample of a STL file. The format only provides a list of independent triangles and their coordinates.

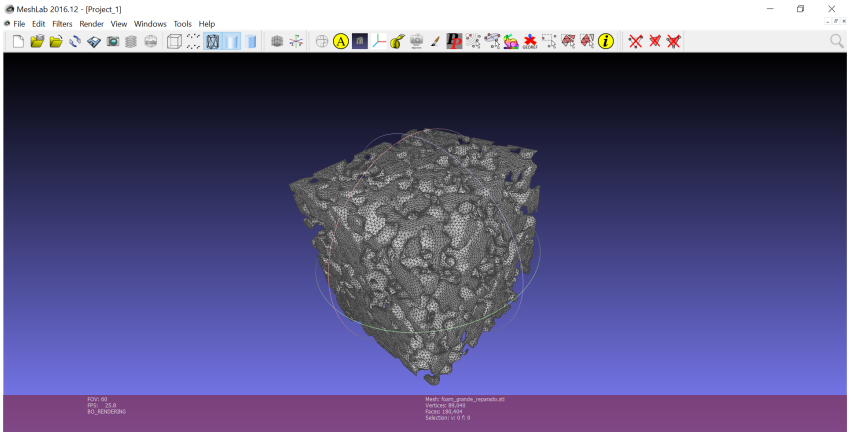

Figure 3.5: Visualisation a STL using Meshlab.

STL editors are also useful to make changes in the geometry. As the file might be huge, it can be convenient to cut the three-dimensional object and obtain a smaller one. This can be done using the "plane cut" tool in Meshmixer. Afterwards it is important to use "make solid" option to be able to mesh the new object. The mesh is built using the "re-mesh" tool which creates a mesh of triangles. Meshmixer allows the user to select some characteristics of the mesh such as regularity or density of the elements. It is also possible to maintain sharp edges.

Once the STL has been repaired and edited, it is possible to obtain a tetrahedral mesh. Again, several editors can perform this task. We have tried Abaqus and Gmsh. The latter has the advantage of being under public licence. In Abaqus, the "Edit Mesh" menu allows to convert an STL into a tetrahedral mesh. Gmsh's interface is less user-friendly but it can perform the task perfectly and seems to be more robust than Abaqus when it comes to big meshes.

Afterwards, Abaqus is used to define the boundaries of the tetrahedral mesh on which conditions will be applied. If the mesh has been produced with Gmsh, it will be necessary to edit the file with the notepad to imitate Abaqus's INP format. This step is required to load it in Abaqus an define the boundaries.

Finally, before using `dolfin-convert` to obtain an XML file, it is necessary to modify other aspects of the original file. On the one hand, the element type must be C3D4 (the linear tetrahedron). As long as the Abaqus mesh is made of tetrahedra of four nodes, one can simply change the name in the INP file (e.g. if the original elements were DC3D4 it is possible to simply rewrite C3D4). On the other hand, when defining a surface, Abaqus writes:

```
*Surface, type = ELEMENT, name = NameOfSurface
```

However, it must be rewritten as follows:

```
*Surface, name = NameOfSurface, type = ELEMENT
```

After using `dolfin-convert`, two files are created. The first one contains the mesh and has the name used in the command. The second has the suffix "_facet_region.xml" and contains the surfaces defined in Abaqus. In this file, surfaces are numbered following their order of appearance in the INP.

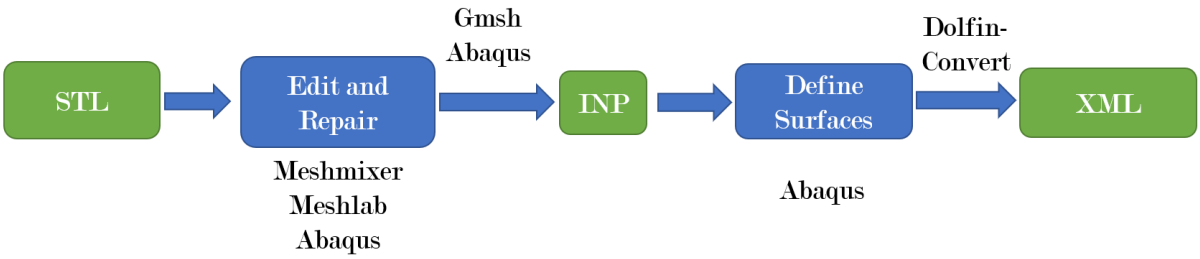

Figure 3.6: Workflow to obtain an XML mesh from an STL file.

Once in FEniCS, the mesh is imported with the following lines:

```
mesh = Mesh('filename.xml') #open the main xml file
facet_regions = MeshFunction('size_t', mesh, 'filename_facet_region.xml')
```

To give an idea of the size of the meshes, the foam scaffold has 38 000 nodes and the structured 45 000.

Type	Number of Nodes
Foam	37 752
Structured	45 542

Table 3.1: Size of the meshes used in the simulations.

3.5.2 Velocity Field

Weak Form

We recall the weak form of the velocity potential:

$$-\int_{\Omega} \nabla\phi\nabla w \; d\Omega = 0 \; . \tag{3.42}$$

Function Space

It is now possible to solve the equation in FEniCS. In the next section we will denote the unknown function ϕ by u, as it is the standard notation for a trial function. The first step is to define a finite element function space using `FunctionSpace`:

```
V = FunctionSpace(mesh, 'P', 1)
```

The type of element and its degree are specified in the second and third arguments, respectively. "P" stands for the standard Lagrange family of elements whereas "1" implies that the element is linear. The element is therefore the so-called "linear triangle". All the different elements supported by FEniCS and their notation can be found on the "Periodic Table of Finite Elements" [2]. More information on this function is provided in the "FEniCS tutorial" (see Langtangen and Logg 2017).

Boundary Conditions

Dirichlet boundary conditions are specified using the following syntax:

```
bcs = []

u_E1 = Expression('x[0]', degree = 1)
```

[2] Available at `http://femtable.org/`.

```
bc_E1 = DirichletBC(V, u_E1, facet_regions, 1)
bcs.append(bc_E1)

u_S1 = Expression('x[0]', degree = 1)
bc_S1 = DirichletBC(V, u_S1, facet_regions, 4)
bcs.append(bc_S1)
```

We first define the expression of the boundary condition. Here it is $u = x$ and we denote x by `x[0]` [3] The second step is to create the `DirichletBC` object. Its arguments are the function space, the expression of the boundary condition, the facet regions file and the facet number.

This lines above would apply a Dirichlet boundary condition (to facet regions number 1 and number 4. The numbering of the facet regions is generated automatically with the `_facet_regions` file, it follows the alphabetical order of the names of the surfaces. We recall that surfaces were manually created using Abaqus (see 3.5.1).

Weak Form in FEniCS

The trial and test functions are created using `TrialFunction` and `TestFunction` and specifying the function space that has been previously defined (here `V`). The right-hand side term of the weak form (i.e. `a`) is written using FEniCS operators `dot` (scalar product between vectors) and `grad` (gradient of a scalar function). The integral is denoted by `dx`. The left-hand side term (i.e. `L`) has to be defined despite being zero in the present case. As it can be seen below, this is done by setting a constant function f.

```
u = TrialFunction(V)
w = TestFunction(V)
f = Constant(0.0)

a = -dot(grad(u),grad(w))*dx
L = f*w*dx
```

Solving (finally) the PDE

Solving the PDE comes down to solving a linear system where the values of u at the nodes of the mesh are the unknowns. The matrices of the system are easily assembled using `assemble()`. Dirichlet boundary conditions have to be taken into account before solving the system. A loop structure is used in the code to go through all of them. Finally, it must be pointed out that, before solving the system, u must be redefined as a function and not a trial function.

```
u = Function(V) #Redefine u as a function
A = assemble(a)
b = assemble(L)

#use the following structure
#for several Dirichlet conditions
for bc in bcs:
bc.apply(A, b)

#Sove the system
solve(A,u.vector(),b)
```

Plotting the Solution

Once the system has been solved, the velocity field is obtained by calculating the gradient of u with the operator `grad`. It can then be plotted with a colourbar using the `plot()` and

[3] y and z would be denoted by `x[1]` and `x[2]` , respectively.

`colorbar()` commands from `matlpotlib` and `matplotlib.pyplot` [4].

```
v = grad(u)
p = plot(v)
plt.colorbar(p)
```

Saving the Mesh and Results

The mesh and the velocity potential (i.e. u) are saved to be used in the advection-diffusion code. The mesh is saved in the XML format used by FEniCS. The suffix `.gz` used below indicates that the file is compressed. The following line creates a folder "velfield" with a mesh file called "domain.xml.gz".

```
File('velfield/domain.xml.gz') << mesh
```

3.5.3 Advection-Diffusion

Opening the Mesh and the Velocity Potential

The mesh is easily opened using the `Mesh` function and specifying the path of the XML file. On the other hand, reading the file containing the velocity potential requires more steps. First of all, the file is opened using `HDF5File`. This function was previously used to create this very same file (see previous section). However, we must now use the reading mode which is activated by `'r'`. Before actually reading the velocity potential, we must specify the we want to read a function. This is done by defining a function u. The values will be stored in this FEniCS object. It is then possible to read the function and finally close the HDF5 file. The velocity potential which was stored as `'u'` is renamed `'phi'` to avoid confusions in the rest of the code.

```
mesh = Mesh('velfield/domain.xml.gz')

V = FunctionSpace(mesh, 'P', 1)
#P1 is linear(1) lagrange(P) element

f = HDF5File(mesh.mpi_comm(),'velpot','r')
u = Function(V)
f.read(u,'u')
f.close()

phi = u #store the potential in variable phi
```

Boundary Conditions

Boundary conditions are specified with the same syntax that was used for the velocity field.

```
bcs =[]

u_W = Expression('1', degree = 1)
bc_W = DirichletBC(V, u_W, facet_regions, 7)
bcs.append(bc_W)

u_E1 = Expression('0', degree = 1)
bc_E1 = DirichletBC(V, u_E1, facet_regions, 1) #zero concentration on the
    inflow face
bcs.append(bc_E1)
```

[4] `matlpotlib` is Python library to build graphs and plots. It is not related to FEniCS.

Weak Form

The FEniCS syntax is very flexible and allows to implement the transient state problem easily. The velocity field is obtained by multiplying the gradient of the velocity potential by 100. As we have seen, the boundary conditions to calculate ϕ where chosen so that the magnitude of the gradient would be close to one (see Section 2.2.1). Therefore, we can multiply it by a scalar to obtain a velocity field with vectors of a chosen magnitude. Given that the units of the mesh are tens of μm, we multiply by 100 to obtain a velocity field of 1000 μ/s.

The expression of the Galerkin variational problem shown in equation 3.27 is written using the FEniCS operator `dot()` and the integral `dx`. The SUPG term is also re-written and added to the variational problem. As defining the right-hand side and the left-hand side terms might be cumbersome, it is possible to store the whole expression in an object named `F` and then use the functions `lhs` and `rhs` to retrive them.

```
#Expressions in variational forms
At = Constant(dt)
mu = Constant(mu)

vel = 100*grad(phi) #velocity field

u = TrialFunction(V)
u_n = Function(V)
v = TestFunction(V)
f = Constant(0.0)

#Residual
r = (u-u_n)/At + dot(vel,grad(u)) - mu*div(grad(u))

#Galerkin Variational Problem
F = dot((u-u_n) / At, v)*dx + dot(vel, grad(u))*v*dx + mu*dot(grad(u), grad(v))
    *dx

#Add SUPG
vnorm = sqrt(dot(vel,vel))
tau = 1/(2*vnorm/h + 4*mu/(h**2)) #tau Codina
F += tau*dot(vel,grad(v))*r*dx

#Identify LHS and RHS
a = lhs(F)
L = rhs(F)
```

Solving the System

As stated in the previous sections, with the transient scheme, the PDE is solved at every time step using the previous solution. A `while` loop is implemented to stop the simulation when the variation between successive solutions goes below a certain threshold. This structure aims at stopping the process when it reaches "stationary conditions". In the code, the threshold received the name `var` for variation. In addition, a variable `t` was used to store the time in seconds of each time step. We wanted to be able to assign a time to every stored solution. For capacity reasons, solutions were only saved every five steps using counters `i` and `count`.

```
#Create VTK to visualise the results
vtkfile_u = File('OUTPUTS/VTK/u.pvd')

#Time-stepping
u = Function(V)

t = 0 #initial time
count = 4 #count to save the solution every 5 steps
```

```
i = 0 #count the number of steps

time = np.array([]) #array to save time steps
var = 1 #var is the difference between succesive solutions

while var > 0.001:

        #Update time
        t = t + dt

        #Solve variational problem for time step
        solve(a == L,u,bcs,solver_parameters={'linear_solver': 'gmres','
            preconditioner': 'ilu'})

        #Save file
        count = count + 1

        if count == 5:
        i = str(i)
        s = u.vector().get_local()
        #b = s > 0.6
        #b = b.astype(np.int)
        np.save('OUTPUTS/VECTORS/vector'+i, s)
        i = int(i)

        time = np.append(time, t)

        vtkfile_u << (u, t)

        i = i + 1
        count = 0


        #difference between successive solutions
        var = u.vector().get_local() - u_n.vector().get_local()
        var = np.amax(var)
        var = abs(var)

        #Update previous solution
        u_n.assign(u)
```

Extracting the Results

This last step is done in order to get a vector with the values of u at the nodes. FEniCS uses a special object called "function" which is different from the vector obtained by solving the linear system. However, it contains the values of the vector. These can be extracted by using the suffix `.vector()`. They are then stored in an array using `.get_local()`. Finally, the values of the array are sorted to match the numbering of the nodes of the mesh.

```
#Get values U
nodal_values_u = u.vector() #intermediate step
array_u = nodal_values_u.get_local()
vertex_values_u = u.compute_vertex_values() #ordered

#Save as VTK
vtkfile = File('FOAM_COARSE/convisual.pvd')
vtkfile << u
```

3.6 Post-Processing of the Results

To compare the performance of the two scaffolds we will measure the concentration of Ca^{2+}. More precisely we will calculate the percentage of fluid volume with a concentration above 0.8. Using the transient state results, we will measure the evolution of this value over time. As the maximum concentration is 1 (see Boundary Conditions), 0.8 is more than three quarters of the maximum volume. It is deemed a sufficiently high value. Volumes can be calculated using the mass matrix seen in Section 3.2.1. This is derived from the following properties of integration.

Using Galerkin's method a function f can be rewritten as:

$$f(\mathbf{x}) = \sum_j N_j(\mathbf{x}) f_j \tag{3.43}$$

Where N_j is a shape function and f_j the value of f at node j. Let us integrate f over its domain Ω:

$$\int_\Omega f(\mathbf{x})\ d\Omega = \sum_j f_j \int_\Omega N_j(\mathbf{x})\ d\Omega \tag{3.44}$$

We introduce vector $\mathbf{p}$ such as:

$$p_j = \int_\Omega N_j(\mathbf{x})\ d\Omega \tag{3.45}$$

Then the integral can be replaced by the scalar product of $\mathbf{f}$ and $\mathbf{p}$

$$\sum_j f_j \int_\Omega N_j(\mathbf{x})\ d\Omega = \mathbf{f}^\top \mathbf{p} \tag{3.46}$$

On the other hand it is known that the coefficients of the mass matrix $\mathbf{M}$ are:

$$M_{ij} = \int_\Omega N_i N_j\ d\Omega \tag{3.47}$$

Which implies that:

$$p_j = \sum_i M_{ij}\ and\ \mathbf{p} = \mathbf{M}\mathbb{1} \tag{3.48}$$

Where $\mathbb{1}$ is a vector of ones. Eventually, the integral is rewritten as:

$$\mathbf{p}^\top \mathbf{f} = \mathbb{1}\mathbf{M}\mathbf{f} \tag{3.49}$$

For $f(\mathbf{x}) = 1$ the integral becomes:

$$\int_\Omega f(\mathbf{x})\ d\Omega = \int_\Omega d\Omega \tag{3.50}$$

Which is the volume of the domain. Therefore using equation 3.49 it is possible to obtain the volume of Ω:

$$V_\Omega = \mathbb{1}\mathbf{M}\mathbb{1} \tag{3.51}$$

This last expression can be seen as a sum of the weights of the nodes. Each one of the j nodes of the domain contributes to the total volume. This can be used to determine the volume of the domain with a concentration above a certain level. By replacing some of the ones by zeros, it is possible to take into account only certain nodes. These nodes are those for which the value of u is above the desired threshold. As stated before, Galerkin's method yields an

algebraic system whose solution $\mathbf{u}$ is a vector of nodal unknowns. Once the vector has been obtained, the coordinates above the threshold are replaced by ones and those below by zeros. Thus a vector $\mathbf{u}'$ is obtained. Eventually, the volume of Ω with a concentration above the threshold is computed by:

$$V = \mathbb{1}\mathbf{M}\mathbf{u}' \tag{3.52}$$

Chapter 4

Results

4.1 Velocity Potential

Figures 4.1 and 4.2 show the velocity potential for the foam and structured scaffolds, respectively. The potential only varies along the X-axis. To see the other velocity potentials used in the study see Appendix B.

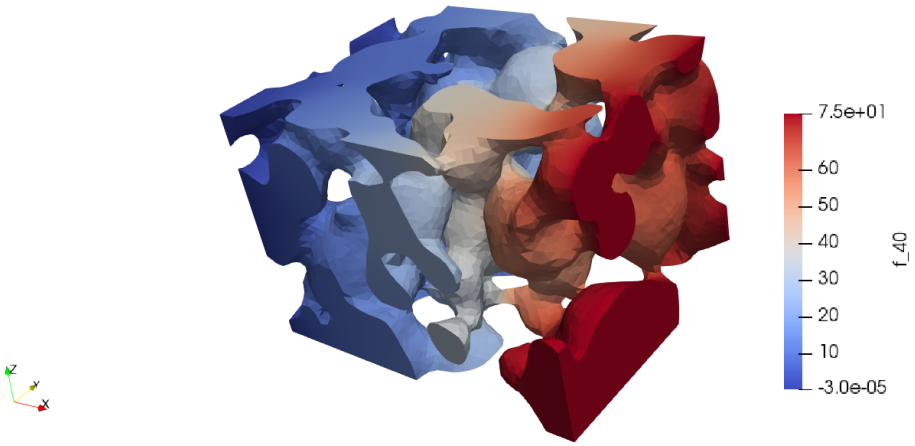

Figure 4.1: Velocity potential in the foam scaffold. Note the variation along the X-axis.

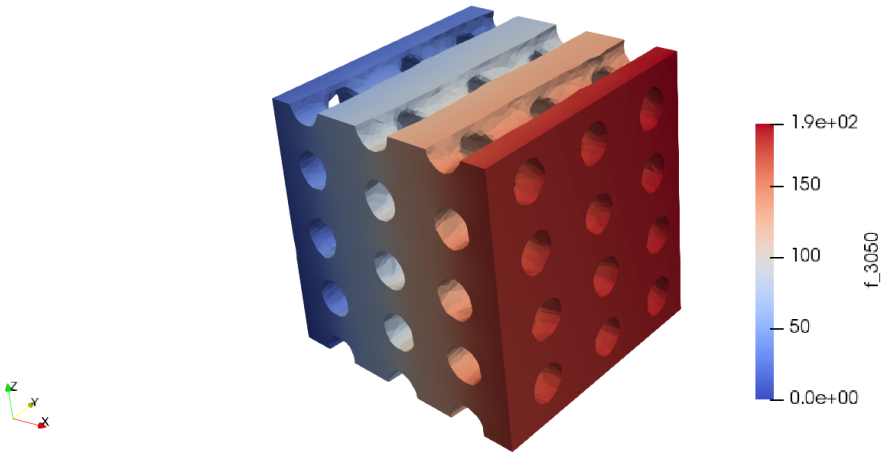

Figure 4.2: Velocity potential in the structured scaffold. Note the variation along the X-axis.

4.2 Steady State

4.2.1 Foam Scaffold

The results of the steady state solution for the foam scaffold are shown below. Figures 4.3, 4.4 and 4.5 are different views of the same solution, where the fluid runs parallel to the X-axis. It is possible to see that the concentration reaches the maximum values in most of the fluid.

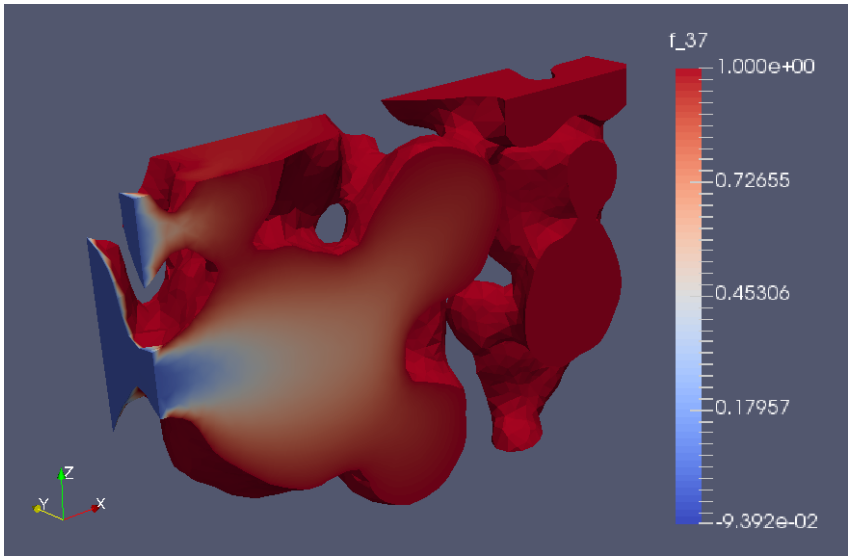

Figure 4.3: Steady state solution for the foam scaffold. The Y-axis is normal to the plane cut while the flow is parallel to the X-axis.

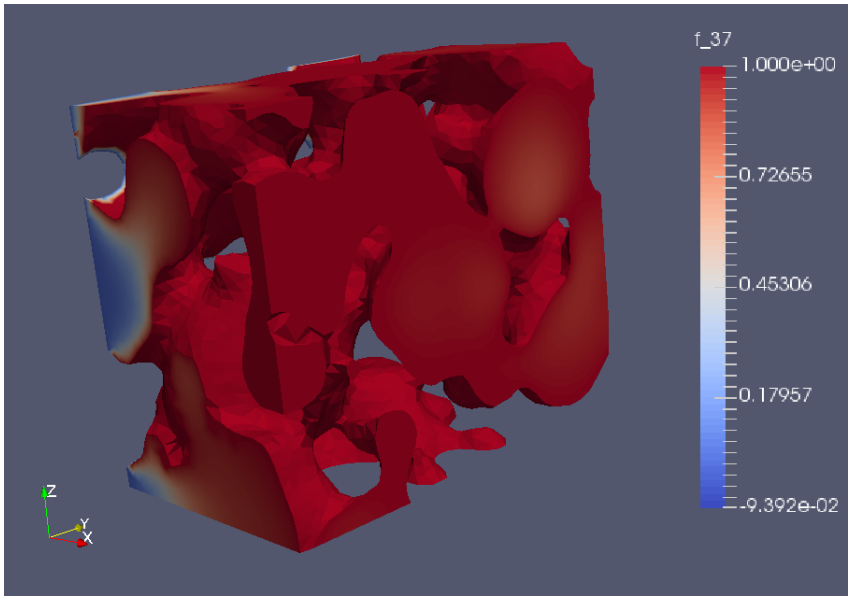

Figure 4.4: Steady state solution for the foam scaffold. The X-axis is normal to the plane cut while the flow is parallel to the X-axis.

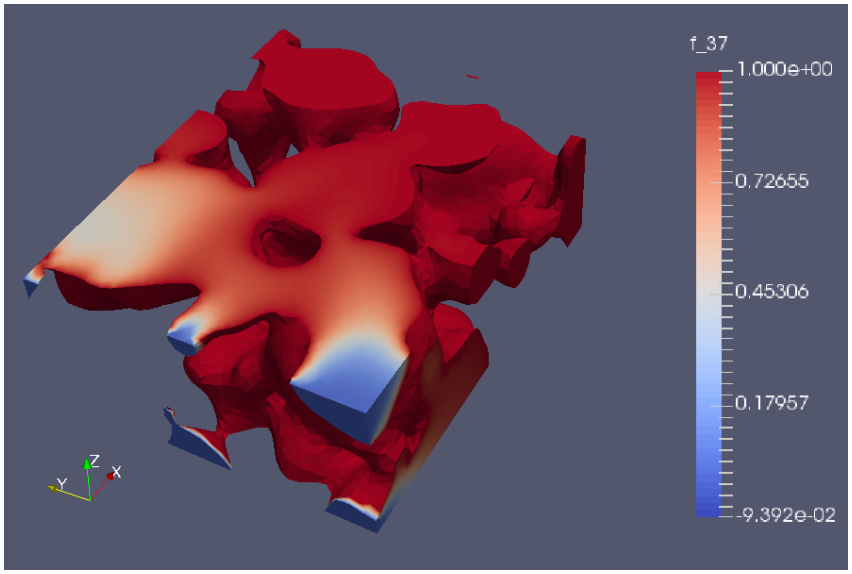

Figure 4.5: Steady state solution for the foam scaffold. The Z-axis is normal to the plane cut while the flow is parallel to the X-axis.

4.2.2 Structured Scaffold

In the structured scaffold, it is possible to see areas with lower concentrations (i.e. lighter shades of red) like in Figure 4.7. This seems to match the initial expectations. Figures 4.6, 4.7 and 4.8 show different views of the same solution. As in the previous section, the flow runs along the X-axis.

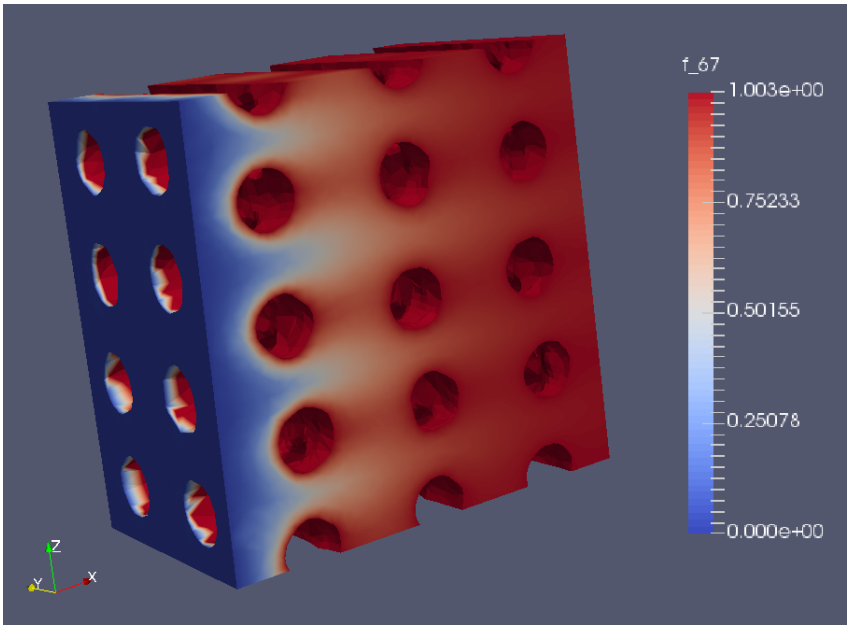

Figure 4.6: Steady state solution for the structured scaffold. The Y-axis is normal to the plane cut while the flow is parallel to the X-axis.

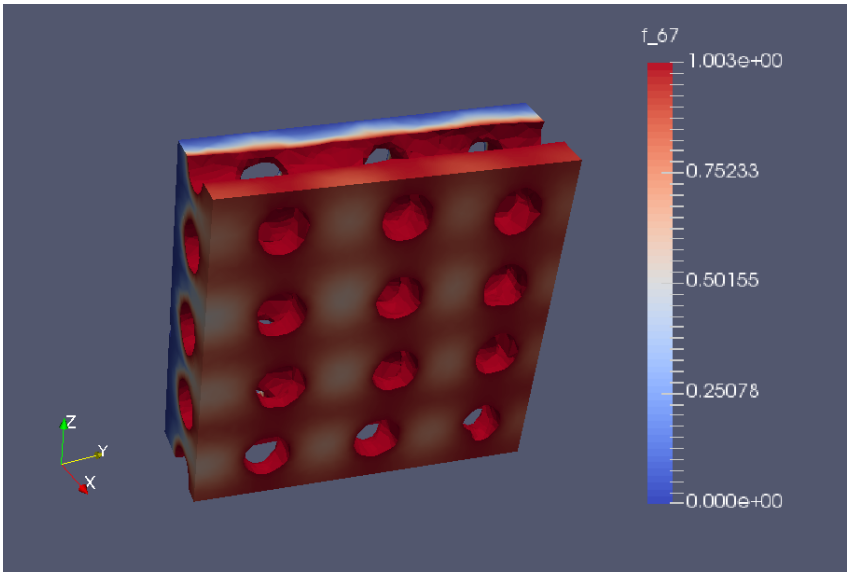

Figure 4.7: Steady state solution for the structured scaffold. The X-axis is normal to the plane cut while the flow is parallel to the X-axis.

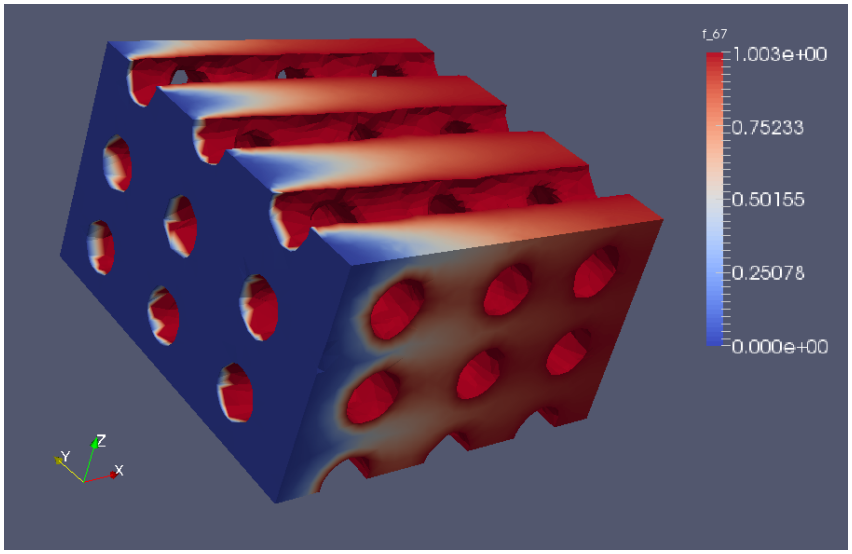

Figure 4.8: Steady state solution for the structured scaffold. The Z-axis is normal to the plane cut while the flow is parallel to the X-axis.

4.3 Transient State

4.3.1 Foam Scaffold

The different stages of the transient state presented below show how the fluid is filled with ions (Figures 4.9, 4.10 and 4.11). In the final stage it is possible to see that the maximum concentration is reached almost everywhere. Only the parts close to the "entrance" show lighter shades of red (i.e. lower concentrations). In these areas ions are easily dragged by the fluid and moved to other parts of the volume.

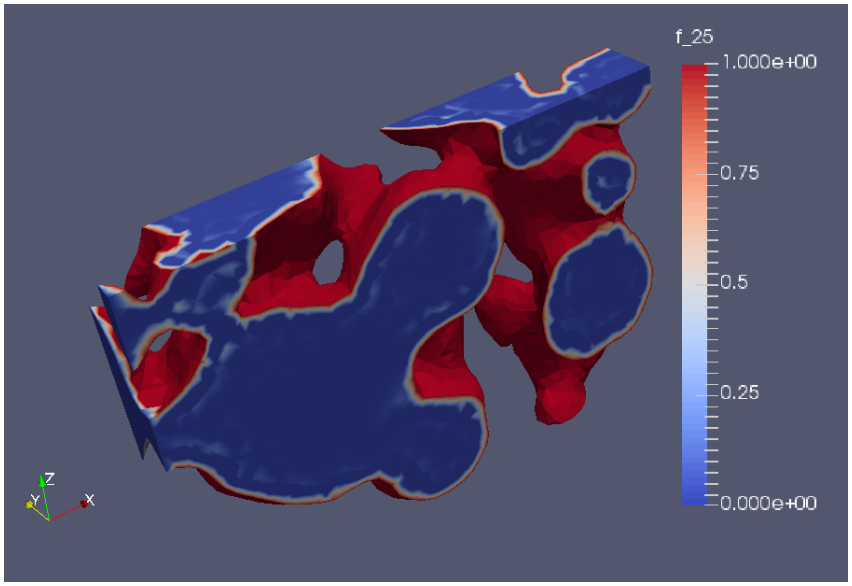

Figure 4.9: Transient state solution for the foam scaffold (initial stage). The Y-axis is normal to the plane cut while the flow is parallel to the X-axis.

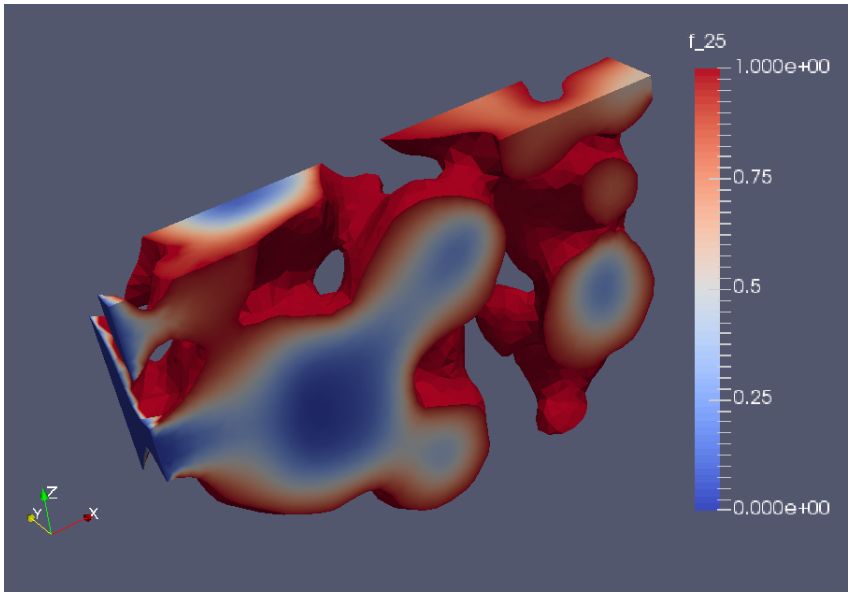

Figure 4.10: Transient state solution for the foam scaffold (intermediate stage). The X-axis is normal to the plane cut while the flow is parallel to the X-axis.

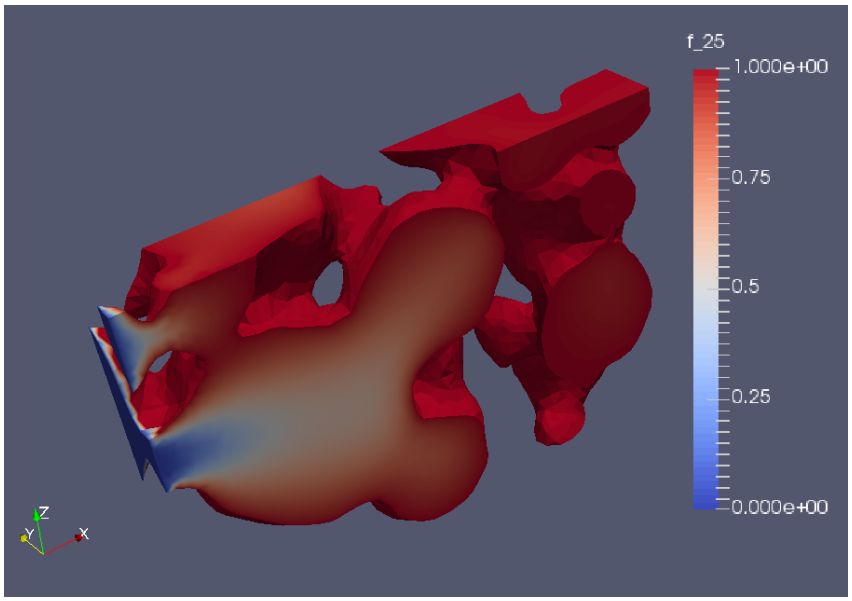

Figure 4.11: Transient state solution for the foam scaffold (final stage). The Z-axis is normal to the plane cut while the flow is parallel to the X-axis.

4.3.2 Structured Scaffold

The following Figures 4.12, 4.13 and 4.14 show three stages of the transient simulation for the structured scffold As for the structured scaffold, the final stage of the transient analysis shows large portions of the volume with very low concentrations (Figure 4.14). It seems that the scaffold's structure constitutes a series of perfect corridors through which the ions are dragged. Therefore it is difficult to find areas where they accumulate, except for the creases at the junction of orthogonal filaments of the scaffold.

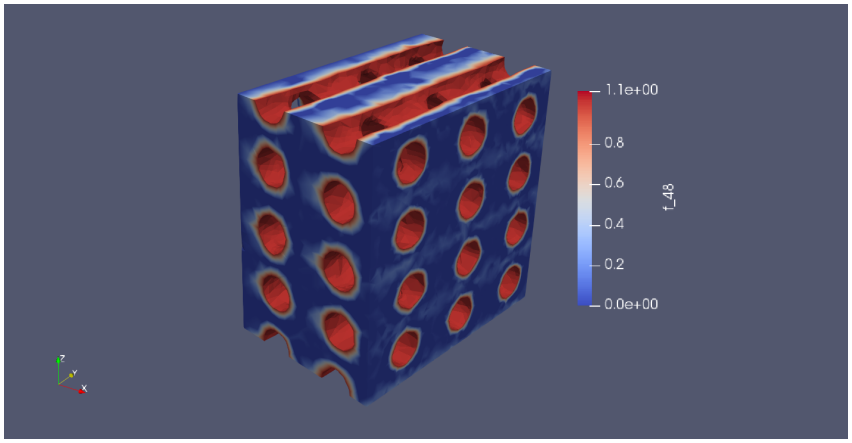

Figure 4.12: Transient state solution for the structured scaffold (initial stage). The flow is parallel to the X-axis.

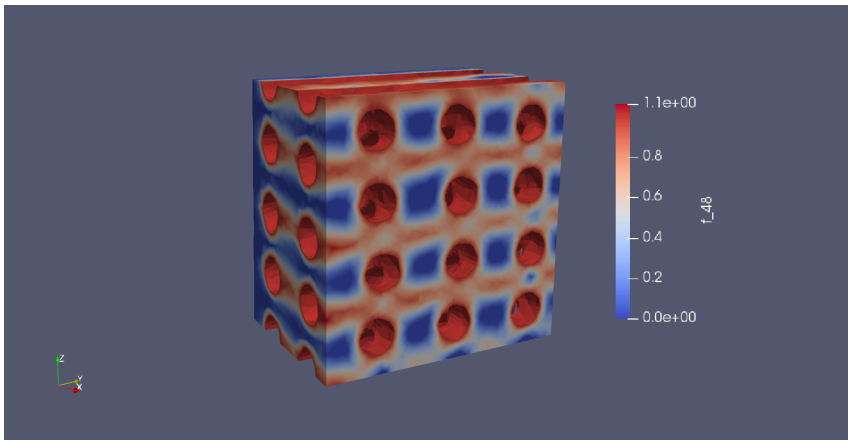

Figure 4.13: Transient state solution for the structured scaffold (intermediate stage). The flow is parallel to the X-axis.

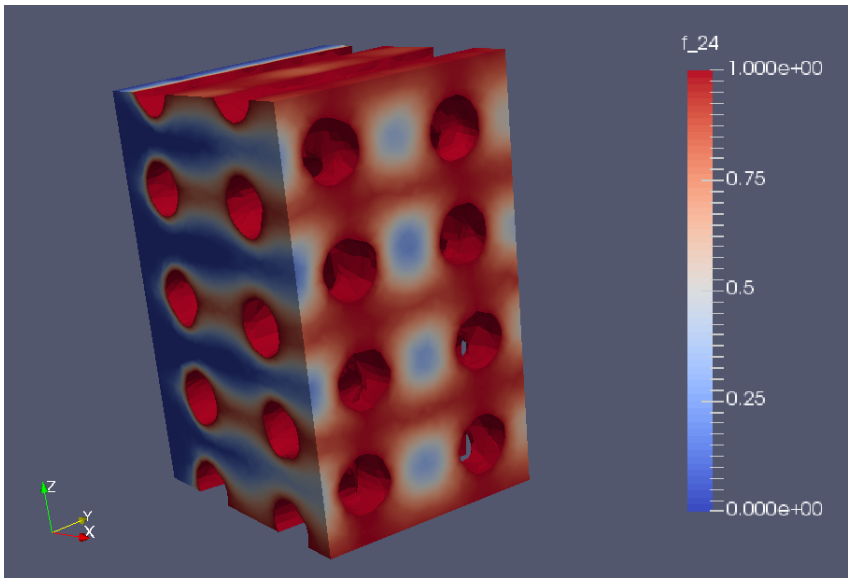

Figure 4.14: Transient state solution for the structured scaffold (final stage). The flow is parallel to the X-axis.

Chapter 5

Analysis of the Results

5.1 Qualitative Analysis

We now wish to compare the simulations with the experimental results obtained by Barba et al. (2017). These results are presented in Figure 5.1, which shows a foam scaffold 12 weeks after implantation. It is possible to see how new bone (grey areas) has appeared inside the pores of the scaffold (pores are black and white areas correspond to the scaffold). One can distinguish three patterns regarding the formation of new bone (marked by A, B and C). First of all, in some pores the layer of newly-formed bone is thicker and uniformly distributed around the walls of the scaffold (A). Secondly, the layer becomes very thin (or does not form) in bottlenecks (B). This is also visible in Figure 5.2 where concentrations diminish in those areas. Finally, some pores present an irregular bone layer (C). Looking at the pore on the right of Figure 5.2, it seems that the layer with the highest concentrations does not have a uniform thickness. Therefore, the newly-formed layers seem to be related with the concentration patterns obtained in the numerical simulation.

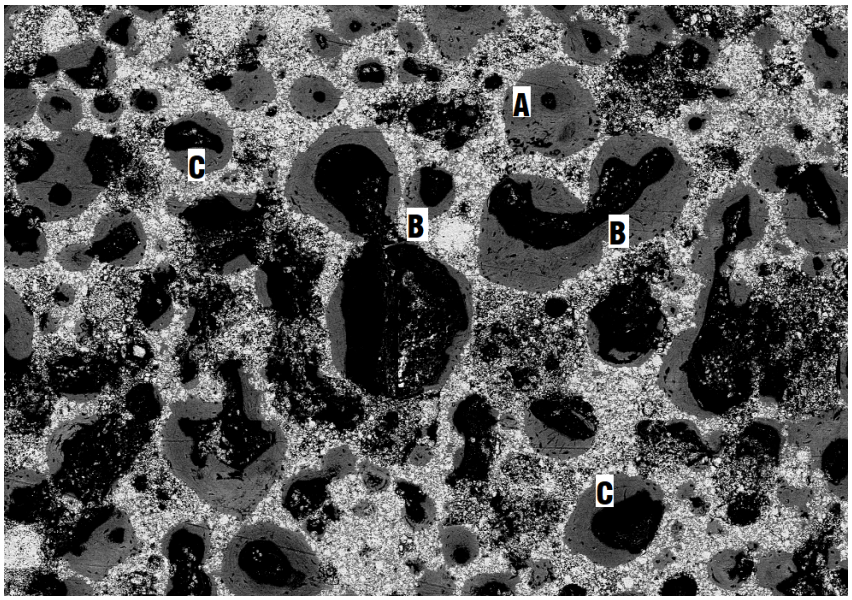

Figure 5.1: Experimental results obtained by Barba, Diez-Escudero, et al. 2017. Black areas correspond to pores (where the interstitial fluid flows), grey to newly-formed bone and white to the scaffold.

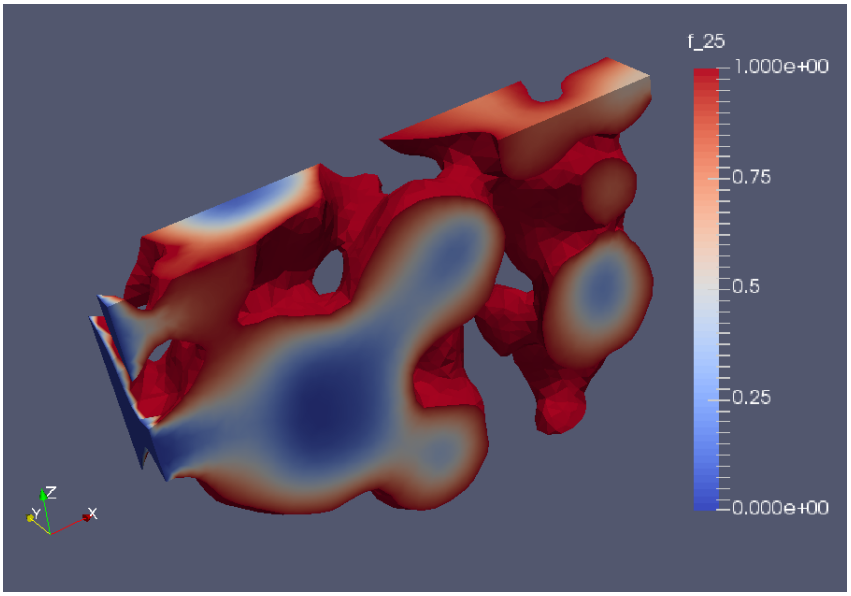

Figure 5.2: Transient state solution for the foam scaffold (intermediate stage). The plane cut is normal to the Y-axis while the flow is parallel to the X-axis.

Regarding the structured scaffold, the simulation also seems to match the experimental results. In 2017, Barba et al. noticed that bone only formed at the intersection of the "filaments" [1] of the scaffold (Figure 5.4). It is only in those areas that some "geometrical irregularity" is visible. In the simulation (Figure 5.4), it is possible to notice that higher concentrations are located at the intersections.

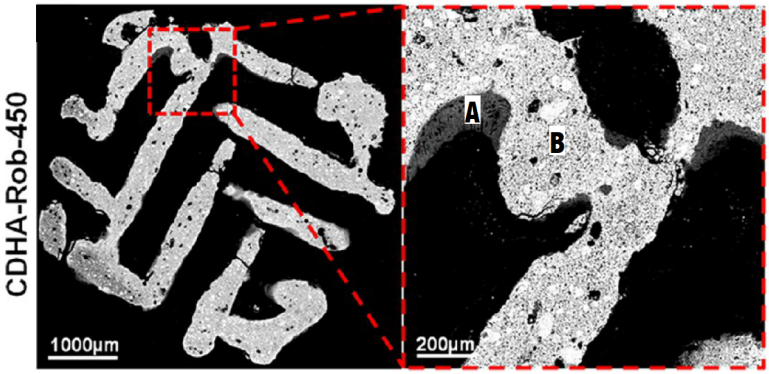

Figure 5.3: Results obtained by Barba et al. (see Barba, Diez-Escudero, et al. 2017) in a structured scaffold. Bone (A) only forms at the intersection of the "filaments" of the scaffold (B).

[1] By filaments we mean the parallel and orthogonal cylinder that make the scaffold.

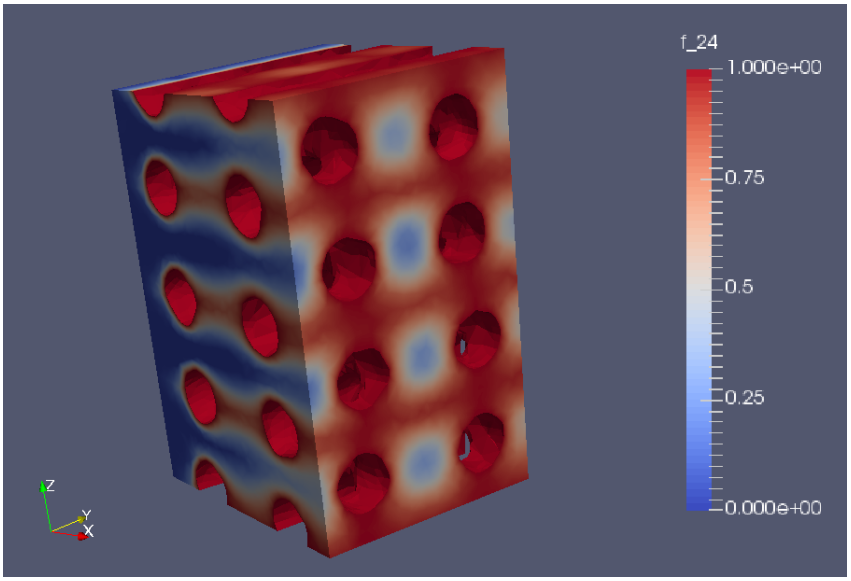

Figure 5.4: Final state of the transient simulation. The highest concentrations are located at the intersection of the "filaments".

5.2 Quantitative Analysis

As stated in Section 3.6, in order to compare both scaffolds, we shall calculate the volume of fluid with a concentration above 0.8. We recall that the simulations have been conducted with three different velocity fields. They are defined below.

- Case 1: Flow along X-axis.
- Case 2: Flow resulting from the addition of a flow along the X-axis and a flow along the Y-axis.
- Case 3: Flow resulting from the addition of a flow along the X-axis, a flow along the Y-axis and a flow along the Z-axis.

For Cases 2 and 3, vectors resulting from the addition of several flows have been normalized in order to be similar in size to those of Case 1.

First, we will compare the two scaffolds under Case 1 and then, the same scaffold under the three cases.

5.2.1 Case 1

The evolution of the concentration (which has been normalized between 0 and 1) is plotted in Figure 5.5. It can be noticed that in the foam scaffold the concentration of ions increases at a higher rate and reaches higher values.

5.2.2 Several flows

Each scaffold has been tested with different velocity fields. The results for the two scaffolds are displayed in Figures 5.6 and 5.7.

The two charts show that concentrations are still higher in the foam scaffold (more than 70% of the volume has a concentration above 0.8). However, it is interesting to see the influence of the different flows. For the foam scaffold, although the three curves have similar shapes, Case 1 stands out as it yields the best results (almost 80% of the volume has a concentration above 0.8). Cases 2 and 3 are very similar and reach a little more than 70% of the volume.

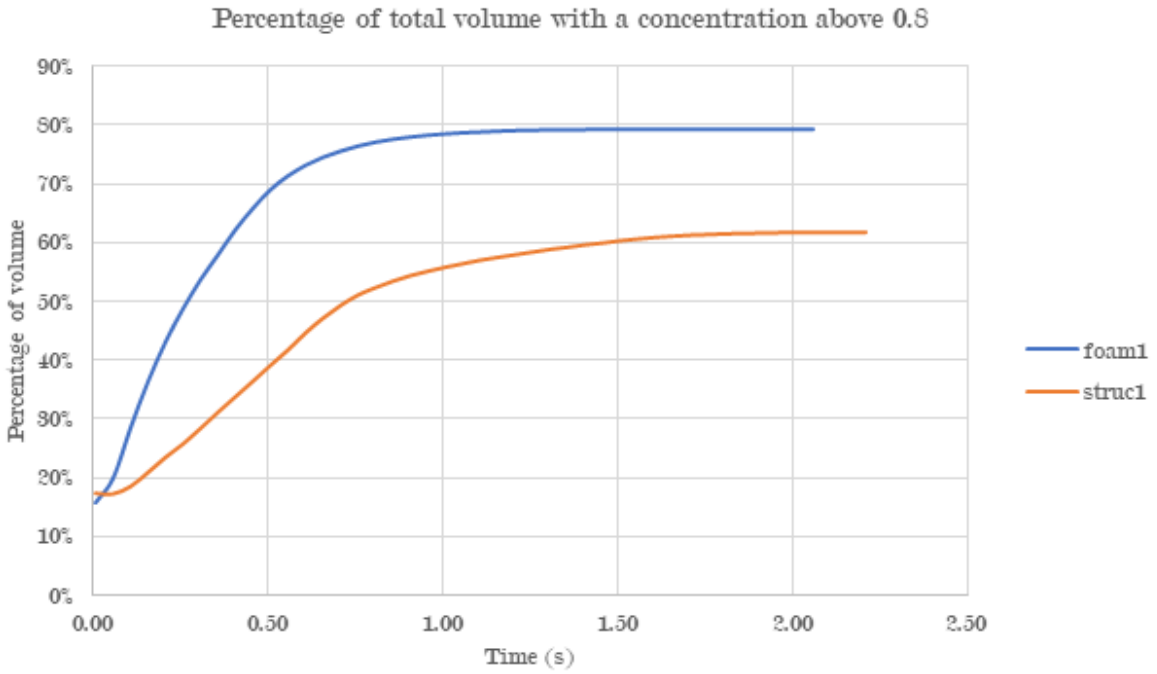

Figure 5.5: Evolution of the Concentration in the scaffolds

Regarding the structured scaffold, the curves have different shapes. Surprisingly, the best results are attained by Case 2. It looks as if by the addition of two flows one is cancelled by the other. In Case 1, the interstitial fluid flows in one direction, caressing the filaments of the scaffold and dragging the ions. With the addition of an orthogonal flow it looks as if the gentle flow of Case 1 is disturbed. The resulting flow is more "chaotic" and ions remain in the scaffold. With the third flow, one would expect higher concentrations across the volume of fluid. However, the results are very similar to those of Case 1. This could be caused by the addition of the third flow which is out of plane.

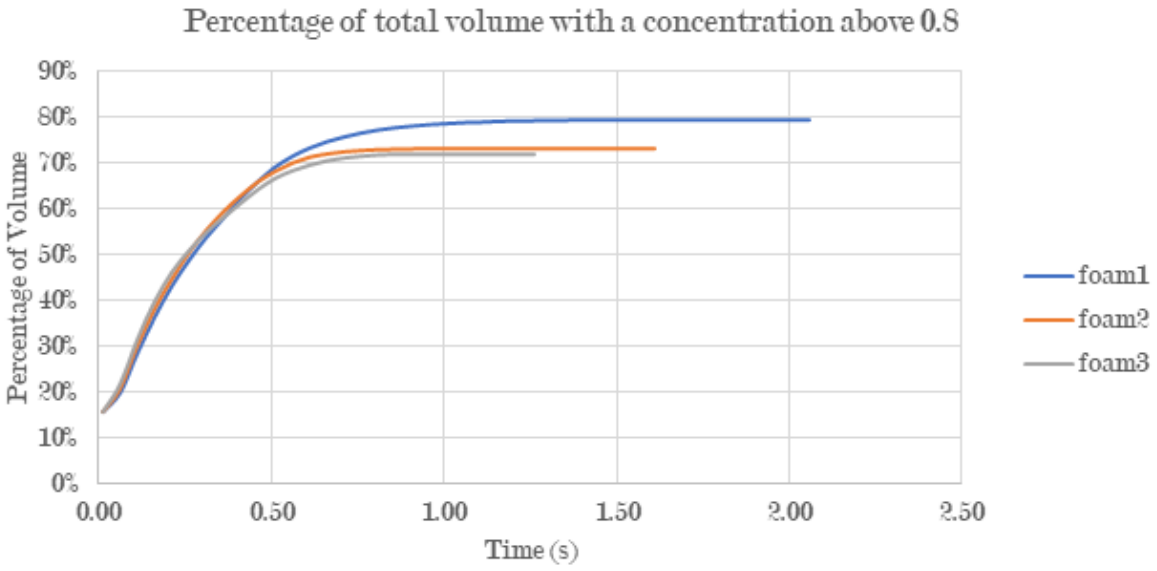

Figure 5.6: Evolution of the Concentration in the Foam Scaffold for Different Flows.

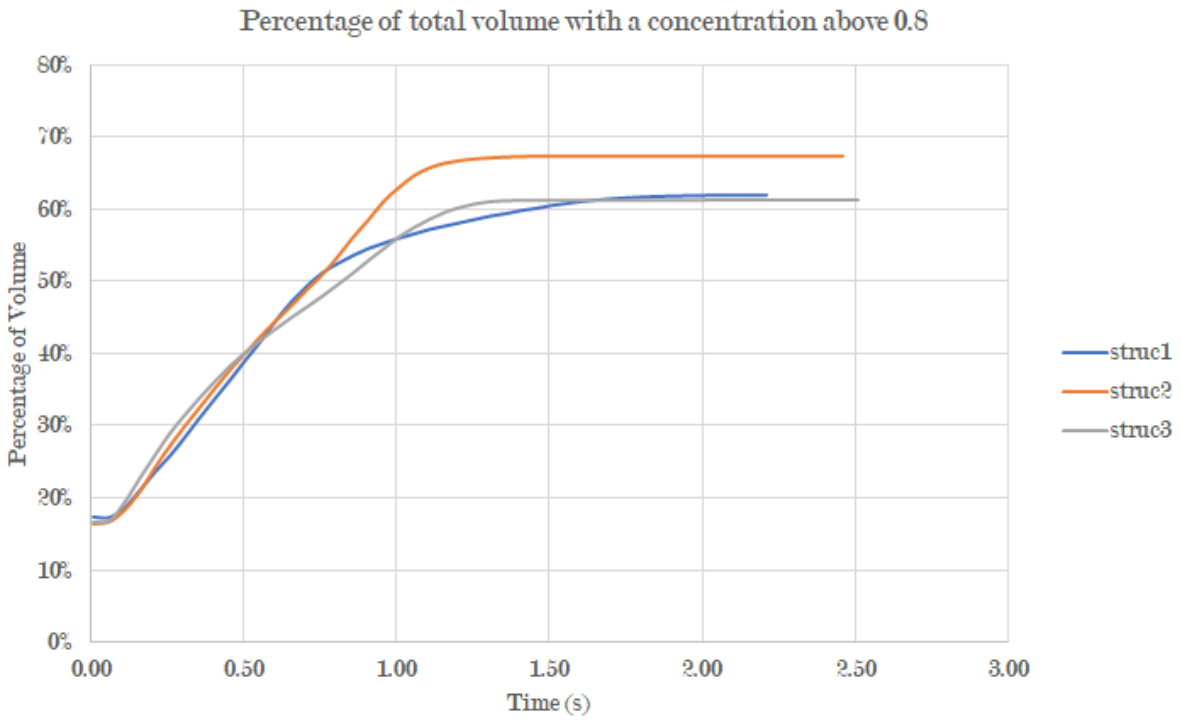

Figure 5.7: Evolution of the Concentration in the Structured Scaffold for Different Flows.

Chapter 6

Conclusions and Future Work

The present study has put forward a simple model to study the influence of the geometry on the formation of bone tissue in scaffolds. More precisely, it has sought to obtain the evolution of Ca^{2+} ions which are thought to be one of the main factors for cell differentiation. Higher concentration implies higher differentiation and therefore more bone formation. As we have seen, the model relies on the advection-diffusion equation. The advection term of this equation requires a velocity field which has been modelled by a potential flow. The potential flow was mainly chosen for its simplicity. The equation has been solved using the Finite Element Method implemented with FEniCS, an open-source platform for solving PDEs.

FEniCS has proved to be a very interesting tool, which could be described as user-friendly and easy to learn. Its simplicity and flexibility enable a great degree of customisation of the code. For example, we were able to implement the transient state analysis very easily. In addition, there is a great amount of documentation available and a wide community on the internet. This is certainly of great help to overcome any difficulty with the code.

During the implementation of the FEM, the major obstacle has been the preparation of the meshes. Meshes had to be created from STL files which are usually poorly suited for the purpose. They contain some flaws that make difficult their conversion to other formats such as INP or XML. Therefore, the process of obtaining the meshes has been rather arduous, if not cumbersome. As explained in Section 3.5.1, several programs were required to carry out the task.

The equation was eventually solved successfully and the solution seems to match the experimental results. On the one hand, concentrations in the foam scaffold were higher, thus suggesting that more differentiation could take place in that scaffold. Conversely, the ordered structure of the 3D-printed scaffold seemed less suited to maintain high concentrations of ions. It is likely that its "corridors" enable the fluid to drag the ions. Finally, examining the behaviour of the scaffold under different velocity fields provided interesting insights. First, we tested a single flow along one axis, then two and three orthogonal flows. In the case of the foam scaffold, the concentration diminshed with the number of flows. As for the structured scaffold, the best highest concentrations were achieved with two flows. With respect to the initial objectives of the project, the result are rather satisfactory.

However, we have identified several improvements to the work. First of all, it could be interesting to work with a bigger geometry, which means working with a bigger portion of the scaffold. Such mesh would be more representative of the whole domain. We have already started to go along this path. However, some difficulties were encountered while using FEniCS with a larger mesh (around 400 000 nodes). Finally, the issues were overcome by changing some of the default settings of FEniCS (see Appendix D).

The velocity potential and the advection-diffusion solution are presented in Figures 6.1, 6.2 and 6.3.

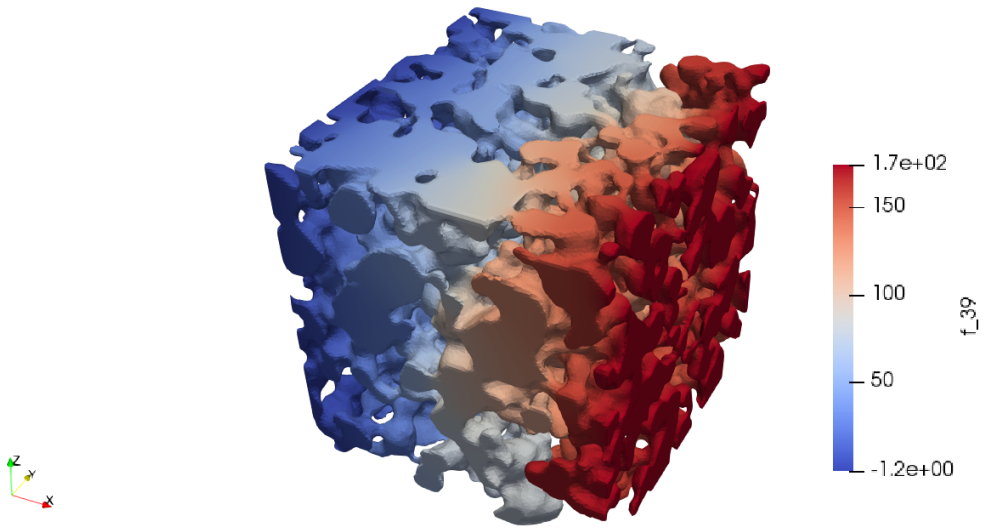

Figure 6.1: Velocity potential for a larger foam scaffold.

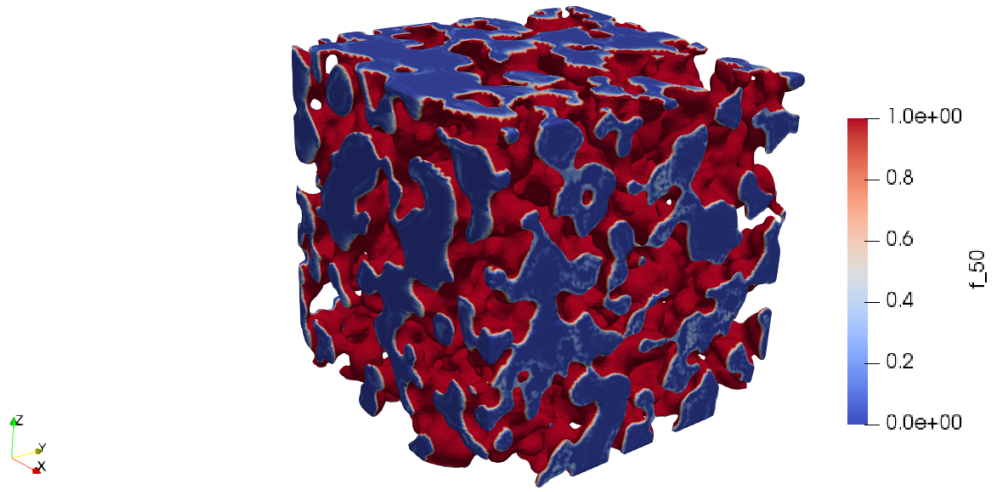

Figure 6.2: Initial state of the larger foam scaffold in a transient analysis.

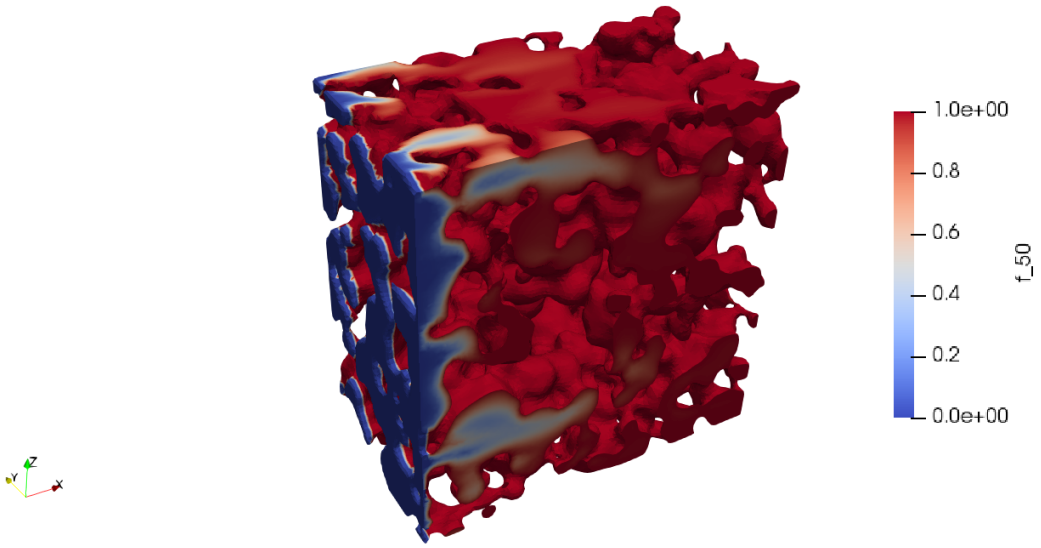

Figure 6.3: Final state of the larger foam scaffold in a transient analysis.

Regarding the velocity field, the potential flow model could be replaced by the Navier-Stokes equations. This would increase the computational cost but could be a more realistic representation of the flow. In addition, it would allow to compute shear stresses, which some suggest foster cell differentiation Yourek et al. 2010.

Finally, the BMPs could be introduced in the model. The evolution of their concentration could be modelled with the advection-diffusion equation with a source term. The value of the source term would depend on the concentration of Ca^{2+} ions.

Appendix A

Function Spaces and Norms

A.1 Function Spaces and Norms

In mathematical terms, the FEM "rests upon the discrete representation of a weak integral form of the partial differential equation to be solved" (Donea and Huerta 2003, p.19). Therefore two steps are required: formulate the weak form and discretise it. In order to achieve it, some function spaces and norms need to be defined. Finite Element functions should possess generalized derivatives and integrability properties (Donea and Huerta 2003, p.20). These characteristics can be found among functions belonging to some Sobolev Spaces.

A.1.1 Sobolev Spaces

First of all, we introduce $\mathcal{L}_2(\Omega)$ the space of functions that are square integrable over the domain Ω. In this space, the inner product is:

$$(u, v) = \int_\Omega uv \; d\Omega \tag{A.1}$$

The norm is:

$$||v||_0 = (v, v)^{1/2} \tag{A.2}$$

$\mathcal{L}_2(\Omega)$ is used to define the Sobolev spaces used in the FEM. A Sobolev space $\mathcal{H}^k(\Omega)$ is defined by a non-negative integer k and an n-tuple $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n) \in \mathbb{N}^n$. Here n stands for the number of space dimensions. Furthermore, we define a non negative integer $|\alpha|$ such that: $|\alpha| = \alpha_1 + \alpha_2 + ... + \alpha_n$.

It is then possible to express a Sobolev space such as:

$$\mathcal{H}^k(\Omega) = \left\{ u \in \mathcal{L}_2(\Omega) \mid \frac{\partial^{|\alpha|} u}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} ... \partial x_n^{\alpha_n}} \in \mathcal{L}_2(\Omega) \; \forall |\alpha| \leq k \right\} \tag{A.3}$$

Therefore functions belonging to $\mathcal{H}^k(\Omega)$ are square integrable and their derivatives of order up to order k as well. The norm in such space is:

$$||u||_k = \left(\sum_{s=0}^{k} \sum_{|\alpha|=s} \left\| \frac{\partial^{|\alpha|} u}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} ... \partial x_n^{\alpha_n}} \right\|_0^2 \right)^{1/2} \tag{A.4}$$

The Sobolev spaces used in the following sections are $\mathcal{H}^0$, $\mathcal{H}^1$ and $\mathcal{H}_0^1$. One can notice that in fact $\mathcal{H}^0(\Omega) = \mathcal{L}_2(\Omega)$. $\mathcal{H}^1(\Omega)$ is defined by:

$$\mathcal{H}^1(\Omega) = \left\{ v \in \mathcal{L}_2(\Omega) | \frac{\partial v}{\partial x_i} \in \mathcal{L}_2(\Omega) i = 1, .., n \right\} \tag{A.5}$$

The inner product is:

$$(u, v)_1 = \int_\Omega \left(uv + \sum_{i=1}^{n} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_i} \right) \, d\Omega \tag{A.6}$$

Thus the norm is:

$$\|u\|_1 = \sqrt{(u, u)_1} \tag{A.7}$$

$\mathcal{H}_0^1(\Omega)$ is simply a subspace of $\mathcal{H}^1$ such that its functions vanish on the boundary of Ω, named Γ. Therefore using mathematical notation:

$$\mathcal{H}_0^1 = \{v \in \mathcal{H}^1(\Omega) \mid v = 0 \; on \; \Gamma\} \tag{A.8}$$

A.1.2 Test and Trial Functions

As stated above, the FEM requires the formulation of the so-called weak form. In order to do so, functions belonging to the Sobolev spaces presented in the previous section are needed. Two classes of functions are defined: test and trial functions.

Test functions are square integrable and have first derivatives which are also integrable over Ω. An additional characteristic is that they vanish on the Dirichlet Boundary Γ_D (i.e. the part of the boundary over which u has prescribed values). Test functions are defined as follows:

$$\mathcal{V} = \left\{w \in \mathcal{H}^1(\Omega) | w = 0 \; on \; \Gamma_D\right\} \tag{A.9}$$

Trial functions are similar to test functions. However they must satisfy the Dirichlet boundary conditions on Γ_D (i.e. the prescribed values). Trial functions are defined as:

$$\mathcal{S} = \left\{u \in \mathcal{H}^1(\Omega) | u = u_D \; on \; \Gamma_D\right\} \equiv \mathcal{V} + \{\bar{u}_D\} \tag{A.10}$$

Where $\bar{u}_D$ is any function in $\mathcal{H}^1(\Omega)$ such that $\bar{u}_D = u_D$. If the Dirichlet conditions are homogeneous (i.e. $u_D = 0$) then $\mathcal{V} = \mathcal{S} = \mathcal{H}_0^1$.

Appendix B

Boundary Conditions

B.1 Case 1

In order to obtain a velocity field that flows along the X-axis, the boundary conditions are:

$$\phi(x, y, z) = x \ for \ \Gamma_1 \cup \Gamma_2 \ , \tag{B.1}$$

where Γ_1 and Γ_2 are the two faces whose normals are parallel to the X-axis. The implementation in FEniCS is:

```
bcs=[]

u_E1 = Expression('x[0]', degree = 1)
bc_E1 = DirichletBC(V, u_E1, facet_regions, 1)
bcs.append(bc_E1)

u_S1 = Expression('x[0]', degree = 1)
bc_S1 = DirichletBC(V, u_S1, facet_regions, 4)
bcs.append(bc_S1)
```

Numbers **1** and **4** in **DirichletBC()** indicate the surface on which the condition is applied. The numbering of the surfaces follows the alphabetical order their names. It is automatically generated within the **_facet_regions.xml** file. The names were given manually when defining the surfaces in Abaqus.

Figure B.1 shows the resulting potential.

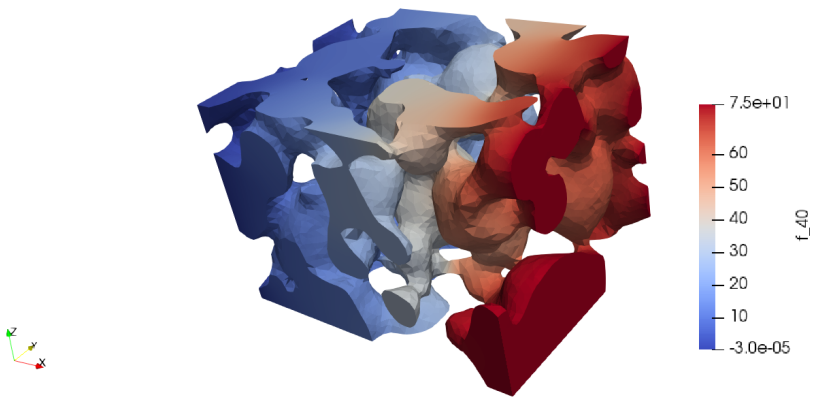

Figure B.1: Velocity potential in the foam scaffold. Note the variation along the X-axis.

B.2 Case 2

For Case 2, we wish to use the flow resulting from the addition of a flow along the X-axis and a flow along the Y-axis. The boundary conditions are then:

$$\phi(x, y, z) = \frac{\sqrt{2}}{2}(x + y) \; for \; \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4 \; , \tag{B.2}$$

where Γ_1 and Γ_2 are the two faces whose normals are parallel to the X-axis. Γ_3 and Γ_4 are the two faces whose normals are parallel to the Y-axis. The code in FEniCS is:

```
bcs = []

u_E1 = Expression('sqrt(2)*0.5*(x[0]+x[1])', degree = 1)
bc_E1 = DirichletBC(V, u_E1, facet_regions, 1)
bcs.append(bc_E1)

u_E2 = Expression('sqrt(2)*0.5*(x[0]+x[1])', degree = 1)
bc_E2 = DirichletBC(V, u_E2, facet_regions, 2)
bcs.append(bc_E2)

u_S1 = Expression('sqrt(2)*0.5*(x[0]+x[1])', degree = 1)
bc_S1 = DirichletBC(V, u_S1, facet_regions, 4)
bcs.append(bc_S1)

u_S2 = Expression('sqrt(2)*0.5*(x[0]+x[1])', degree = 1)
bc_S2 = DirichletBC(V, u_S2, facet_regions, 5)
bcs.append(bc_S2)
```

The resulting potential is shown in Figure B.2.

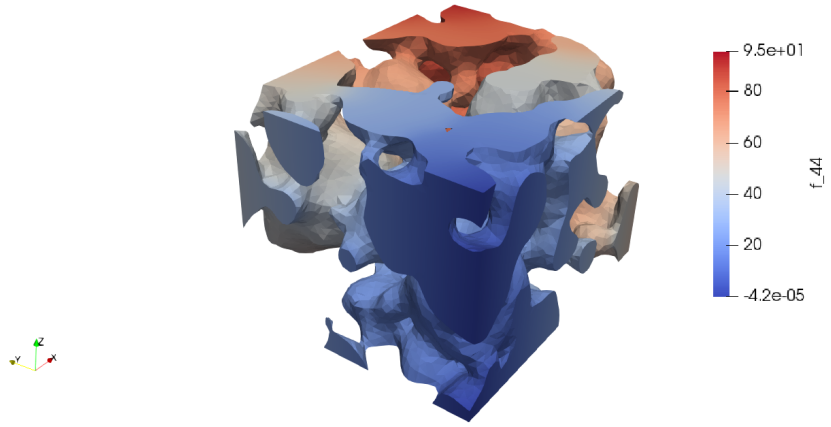

Figure B.2: Velocity potential in the foam scaffold. Note the variation along the X-axis and the Y-axis.

B.3 Case 3

In Case 3, the flow results from the addition of three flows: a flow along the X-axis, a flow along the Y-axis and a flow along the Z-axis. The boundary conditions are:

$$\phi(x, y, z) = \frac{\sqrt{3}}{3}(x + y + z) \; for \; \partial\Omega \; , \tag{B.3}$$

where $\partial\Omega$ is the boundary of the whole domain. To implement the boundary conditions in FEniCS, the Dirichlet condition is applied to each one of the faces of the domain.

```
bcs = []

u_E1 = Expression('(1/sqrt(3))*(x[0]+x[1]+x[2])', degree = 1)
bc_E1 = DirichletBC(V, u_E1, facet_regions, 1)
bcs.append(bc_E1)

u_S1 = Expression('(1/sqrt(3))*(x[0]+x[1]+x[2])', degree = 1)
bc_S1 = DirichletBC(V, u_S1, facet_regions, 4)
bcs.append(bc_S1)

u_E2 = Expression('(1/sqrt(3))*(x[0]+x[1]+x[2])', degree = 1)
bc_E2 = DirichletBC(V, u_E2, facet_regions, 2)
bcs.append(bc_E2)


u_S2 = Expression('(1/sqrt(3))*(x[0]+x[1]+x[2])', degree = 1)
bc_S2 = DirichletBC(V, u_S2, facet_regions, 5)
bcs.append(bc_S2)

u_E3 = Expression('(1/sqrt(3))*(x[0]+x[1]+x[2])', degree = 1)
bc_E3 = DirichletBC(V, u_E3, facet_regions, 3)
bcs.append(bc_E3)

u_S3 = Expression('(1/sqrt(3))*(x[0]+x[1]+x[2])', degree = 1)
bc_S3 = DirichletBC(V, u_S3, facet_regions, 6)
bcs.append(bc_S3)
```

Figure B.3 depicts the resulting potential obtained by solving the PDE.

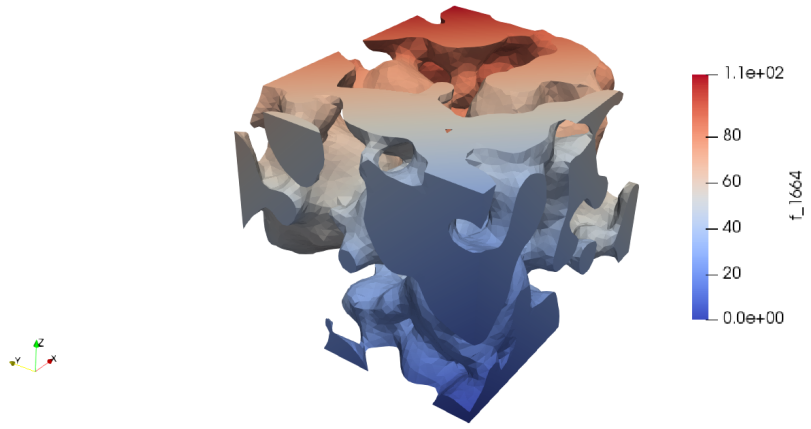

Figure B.3: Velocity potential in the foam scaffold. Note the variation along the three axis.

Appendix C

Complete Codes

C.1 Velocity Field

The following codes yields a velocity potential which varies along the X-axis. To obtain a different velocity field, boundary conditions have to be changes as explained in B.

```
from fenics import *
import matplotlib.pyplot as plt
import matplotlib as mpl
from dolfin import *
from mshr import *
from numpy import *

#Import Mesh
name = 'filename' #name of the xml file without the extension
mesh = Mesh(name + '.xml')
facet_regions = MeshFunction("size_t", mesh, name +"_facet_region.xml")

#Define Function Space
V = FunctionSpace(mesh, 'P', 1)

#Boundaries
bcs = []

u_E1 = Expression('x[0]', degree = 1)
bc_E1 = DirichletBC(V, u_E1, facet_regions, 1)
bcs.append(bc_E1)

u_S1 = Expression('x[0]', degree = 1)
bc_S1 = DirichletBC(V, u_S1, facet_regions, 4)
bcs.append(bc_S1)

#Weak Form
u = TrialFunction(V)
v = TestFunction(V)
f = Constant(0.0)

a = dot(grad(u),grad(v))*dx
L = f*v*dx

#Assemble Matrices
u = Function(V)

A = assemble(a)
b = assemble(L)

#Apply Boundary Conditions
for bc in bcs:
```

```
bc.apply(A, b)

#Solve the System
solve(A,u.vector(),b)

#Save the Solution
Hdf = HDF5File(mesh.mpi_comm(), 'velpot',"w")
Hdf.write(u,'u')
Hdf.close()

#Save VTK files for visualisation

vel = grad(u) #Velocity Field
W = VectorFunctionSpace(mesh,'P', 1)
velvec = project(vel, W)

vtkfile = File('velvisual.pvd')
vtkfile << velvec

vtkfile = File('potvisual.pvd')
vtkfile << u
```

C.2 Advection-Diffusion

C.2.1 Steady State

The following code is used to solve the advection-diffusion equation in steady conditions (i.e. $\frac{\partial u}{\partial t} = 0$).

```
#from fenics import *
import matplotlib.pyplot as plt
import matplotlib as mpl
from dolfin import *
from mshr import *
from numpy import *

#Read mesh from file
name = 'filename' #name of the file
mesh = Mesh(name +'.xml')
facet_regions = MeshFunction("size_t", mesh, name + "_facet_region.xml")

#Define function space
V = FunctionSpace(mesh, 'P', 1) #P1 stands for linear(1) lagrange(P) element

#Import potential
f = HDF5File(mesh.mpi_comm(),'FOAM_COARSE_VEL/velpot','r')
u = Function(V)
f.read(u,'u')
f.close()

phi = u #save potential under name phi

#Boundary Conditions
bcs =[]

u_W = Expression('1', degree = 1)
bc_W = DirichletBC(V, u_W, facet_regions, 7)
bcs.append(bc_W)

u_E1 = Expression('0', degree = 1)
```

```
bc_E1 = DirichletBC(V, u_E1, facet_regions, 1) #zero concentration on the
    inflow face
bcs.append(bc_E1)

#velocity
vel = 100*grad(phi)

#Diffusion Coefficient
mu = 13.3

#Weak Form

u = TrialFunction(V)
v = TestFunction(V)
f = Constant(0.0)

a = (mu*dot(grad(u), grad(v)))*dx + dot(vel,grad(u)*v)*dx
L = f*v*dx

#Assemble Matrices
u = Function(V) # u is redefined (important step)

A = assemble(a)
b = assemble(L)

#Apply BC to Matrices
for bc in bcs:
bc.apply(A, b)

#Solve the System
solve(A,u.vector(),b)

#Get values U
nodal_values_u = u.vector() #intermediate step
array_u = nodal_values_u.get_local()
vertex_values_u = u.compute_vertex_values() #ordered

#Save as VTK
vtkfile = File('FOAM_COARSE/convisual.pvd')
vtkfile << u
```

C.2.2 Transient State

The following code was used to perform a transient state analysis of the advection-diffusion problem. The boundary conditions are set for a flow along the X-axis.

```
from fenics import *
import matplotlib.pyplot as plt
import matplotlib as mpl
from dolfin import *
from mshr import *
import numpy as np
import os as os

#Create a folder to Store the Outputs
os.makedirs('OUTPUTS/VECTORS')

#Data of the problem
dt = 0.01 #Time-step
mu = 13.3 #Diffusion coefficient

#Read mesh from file
```

```
name = 'filename' #name of the file
mesh = Mesh(name +'.xml')
facet_regions = MeshFunction("size_t", mesh, name + "_facet_region.xml")
h = mesh.hmax()

#Open velocity
V = FunctionSpace(mesh, 'P', 1) #P1 is linear(1) lagrange(P) element
f = HDF5File(mesh.mpi_comm(),'velpot','r')
u = Function(V)
f.read(u,'u')
f.close()
phi = u #store velocity potential in phi

#Boundary Conditions
bcs =[]

u_W = Expression('1', degree = 1)
bc_W = DirichletBC(V, u_W, facet_regions, 7)
bcs.append(bc_W)

u_E1 = Expression('0', degree = 1)
bc_E1 = DirichletBC(V, u_E1, facet_regions, 1) #zero concentration on the
    inflow face
bcs.append(bc_E1)

#Expressions in variational forms
At = Constant(dt)
mu = Constant(mu)

vel = 100*grad(phi)

u = TrialFunction(V)
u_n = Function(V)
v = TestFunction(V)
f = Constant(0.0)

#Residual
r = (u-u_n)/At + dot(vel,grad(u)) - mu*div(grad(u))

#Galerkin Variational Problem
F = dot((u-u_n) / At, v)*dx + dot(vel, grad(u))*v*dx + mu*dot(grad(u), grad(v))
    *dx

#Add SUPG
vnorm = sqrt(dot(vel,vel))
tau = 1/(2*vnorm/h + 4*mu/(h**2)) #tau Codina
F += tau*dot(vel,grad(v))*r*dx

a = lhs(F)
L = rhs(F)

#Create VTK to visualise the results
vtkfile_u = File('OUTPUTS/VTK/u.pvd')

#Time-stepping
u = Function(V)

t = 0 #initial time
count = 4 #count to save the solution every 5 steps
i = 0 #count the number of steps

time = np.array([]) #array to save time steps
var = 1 #var is the difference between solutions
```

```
while var > 0.001:

        #Update time
        t = t + dt

        #Solve variational problem for time step
        solve(a == L,u,bcs,solver_parameters={'linear_solver': 'gmres','
            preconditioner': 'ilu'})

        #Save file
        count = count + 1

        if count == 5:
        i = str(i)
        s = u.vector().get_local()
        #b = s > 0.6
        #b = b.astype(np.int)
        np.save('OUTPUTS/VECTORS/vector'+i, s)
        i = int(i)

        time = np.append(time, t)

        vtkfile_u << (u, t)

        i = i + 1
        count = 0


        #difference between successive solutions
        var = u.vector().get_local() - u_n.vector().get_local()
        var = np.amax(var)
        var = abs(var)

        #Update previous solution
        u_n.assign(u)



np.save('OUTPUTS/time',time)
```

Appendix D

Working with Bigger Meshes

We have noticed that FEniCS may experience difficulties when handling big meshes (e.g. 400 000 nodes). The issue seems to be related to the default solver, which is based on Sparse LU Decomposition (Gaussian elimination). This a robust method which works for up to a few thousand unknowns. However, it is not recommended for larger problems as it runs out of memory. Instead, it is preferable to use an iterative solver. Iterative solvers use much less memory and are usually faster [1]. In the following paragraphs we explain how those solvers can be specified in FEniCS within the scope of our project. Regarding the velocity field, once it has

been computed using the regular functions, it must be saved using the following settings. An extra step is required: the FEniCS "gradient object" needs to be projected on vector function space.

```
vel = grad(u) #Velocity Field

#Project the gradient on a Vector Function Space
W = VectorFunctionSpace(mesh,'DG', 0)
velvec = project(vel, W, solver_type='cg', preconditioner_type='amg')

#Save the field to use in advection-diffusion
Hdf = HDF5File(mesh.mpi_comm(), 'velvec.hdf5', 'w')
Hdf.write(velvec,'velvec')
Hdf.close()

#Save for visualisation
vtkfile = File('velvec.pvd')
vtkfile << velvec
```

As the potential was obtained in a linear function space, the field has to be projected on a "piecewise constant function space". The latter is specified using the function `VectorFunctionSpace(mesh,'DG', 0)`, where `DG` stands for Discontinuous Galerkin and `0` stands for the order (i.e. `1` would be linear). We also need to specify the solver and the preconditioner used to do the projection. The Conjugate Gradient solver (abreviated as `cg`) is rather robust and is able to perform the task.

As for the advection-diffusion, the problem seems to lie with the default solver used to obtain the nodal values. Therefore a different solver has to be specified. Like the Conjugated Gradient, the GMRES (Generalized minimal residual method) is also an iterative method. They both belong to the family of the Krylov solvers. It can be called within the `solve()` function in the following manner.

```
solve(A,u.vector(),b,'gmres','amg')
```

[1] More information on solvers can be found on pages 115 to 118 of the FEniCS tutorial (Langtangen and Logg 2017)

Appendix E

Possible Error in Dolfin-Convert

FEniCS version used: Anaconda package 2019.1.0

An error was detected in the `dolfin-convert` script for INP files. We do not know if this error is only found in the Anaconda package of FEniCS. The file is called "abaqus.py" and is located in directory:

```
/home/username/anaconda3/envs/fenicsproject/lib/python3.7/site-packages/
    dolfin_utils/meshconvert/abaqus.py
```

On line 363 one can read:

```
# Now process the facet markers
dim = 2
mesh.init(dim, 0)
facets_as_nodes = mesh.topology()(dim, 0)().reshape(mesh.num_facets(), 3)
```

However it is not possible to use `.reshape()` if the object is not a numpy array. Therefore it is necessary to do the following modifications:

```
facets_as_nodes = np.array(mesh.topology()(dim, 0)()).reshape(mesh.num_facets()
    , 3)
```

The object `mesh.topology()` is defined as a numpy array with `np.array()`.

Eventually, it is possible to convert the Abaqus file to a XML file by typing the following line in the shell:

```
dolfin-convert inputfile.inp outputfile.xml
```

It goes without saying that 'input' and 'output' have to be substituted by the real names of the files.

E.1 Dirichlet Boundary Conditions

In order to apply Dirichlet boundary conditions on a certain region of the mesh (e.g. the face of a cube mesh), it is possible to use Abaqus surfaces. When using dolfin-convert these are stored in an additional XML file and numbered following their order of appearance in the INP file. This additional file is generated automatically with the suffix "_facet_region.xml". It can be read with the following lines:

```
mesh = Mesh('filename.xml') #open the main xml file
facet_regions = MeshFunction('size_t', mesh, 'filename_facet_region.xml')
```

Dirichlet boundary conditions are specified using the same syntax as previously:

```
V = FunctionSpace(mesh, 'P', 1)
u_L = Expression('0', degree = 1)
DirichletBC(V,u_L,facet_regions,1) #apply to facet region 1
```

This lines would apply a Dirichlet boundary condition (here $u = 0$) to facet region number 1, which corresponds to the first surface specified in the INP file.

Bibliography

Barba, Albert, Anna Diez-Escudero, et al. (2017). "Osteoinduction by foamed and 3D-printed calcium phosphate scaffolds: effect of nanostructure and pore architecture". In: *ACS applied materials & interfaces* 9.48, pp. 41722–41736.

Barba, Albert, Yassine Maazouz, et al. (2018). "Osteogenesis by foamed and 3D-printed nanostructured calcium phosphate scaffolds: Effect of pore architecture". In: *Acta biomaterialia* 79, pp. 135–147.

Barradas, A.M. et al. (2011). "Osteoinductive biomaterials: current knowledge of properties, experimental models and biological mechanisms". In: *Eur Cell Mater* 21.407, p. 29.

Beidokhti, Hamid Naghibi et al. (2016). "A comparison between dynamic implicit and explicit finite element simulations of the native knee joint". In: *Medical engineering & physics* 38.10, pp. 1123–1130.

Boccaccio, A et al. (2011). "Finite element method (FEM), mechanobiology and biomimetic scaffolds in bone tissue engineering". In: *International journal of biological sciences* 7.1, p. 112.

Bose, Susmita, Sahar Vahabzadeh, and Amit Bandyopadhyay (2013). "Bone tissue engineering using 3D printing". In: *Materials Today* 16.12, pp. 496–504. ISSN: 1369-7021. DOI: https://doi.org/10.1016/j.mattod.2013.11.017. URL: http://www.sciencedirect.com/science/article/pii/S136970211300401X.

Division of Transplantation (2020). *Organ Donation Statistics*. URL: https://www.organdonor.gov/statistics-stories/statistics.html. (accessed: 10.06.2020).

Donea, J. and A. Huerta (2003). *Finite Element Methods for Flow Problems*. Finite Element Methods for Flow Problems. Wiley. ISBN: 9780471496663. URL: https://books.google.se/books?id=S4URqrTtSXoC.

Dunn, S., A. Constantinides, and P.V. Moghe (2005). *Numerical Methods in Biomedical Engineering*. Biomedical Engineering. Elsevier Science. ISBN: 9780080470801.

Evans, Nicholas D., Eileen Gentleman, and Julia M Polak (2006). "Scaffolds for stem cells". In: *Materials Today* 9.12, pp. 26–33.

Garcia, A.J. and P. Ducheyne (1994). "Numerical analysis of extracellular fluid flow and chemical species transport around and within porous bioactive glass". In: *Journal of biomedical materials research* 28.8, pp. 947–960.

García-González, Alberto et al. (2018). "The Effect of Cell Morphology on the Permeability of the Nuclear Envelope to Diffusive Factors". In: *Frontiers in Physiology* 9, p. 925. ISSN: 1664-042X. DOI: 10.3389/fphys.2018.00925. URL: https://www.frontiersin.org/article/10.3389/fphys.2018.00925.

King, M.R. and N.A. Mody (2010). *Numerical and Statistical Methods for Bioengineering: Applications in MATLAB*. Cambridge Texts in Biomedical Engineering. Cambridge University Press. ISBN: 9781139493420. URL: https://books.google.es/books?id=gEDRKeoIHmcC.

Kojić, M. et al. (2008). *Computer Modeling in Bioengineering: Theoretical Background, Examples and Software*. Wiley. ISBN: 9780470751756.

Lacroix, Damien et al. (2006). "Micro-finite element models of bone tissue-engineering scaffolds". In: *Biomaterials* 27.30, pp. 5326–5334.

Langer, R. and J.P. Vacanti (1993). "Tissue engineering". In: *Science* 260.5110, pp. 920–926. ISSN: 0036-8075. DOI: 10.1126/science.8493529. eprint: https://science.sciencemag.org/content/260/5110/920.full.pdf. URL: https://science.sciencemag.org/content/260/5110/920.

Langtangen, Hans Petter and Anders Logg (2017). *Solving PDEs in Python*. Springer. ISBN: 978-3-319-52461-0. DOI: 10.1007/978-3-319-52462-7.

Ministerio de Sanidad (2019). *Nota de Prensa 28-08-2019*. URL: http://www.ont.es/prensa/NotasDePrensa/28%2008%202019%20%20REGISTRO%20MUNDIAL%20DE%20TRASPLANTES.pdf. (accessed: 10.06.2020).

Organización Nacional de Transplantes (2019). *Actividad de Donación y Transplante*. URL: http://www.ont.es/infesp/Memorias/ACTIVIDAD%20DE%20DONACI%C3%93N%20Y%20TRASPLANTE%20ESPA%C3%91A%202019.pdf. (accessed: 10.06.2020).

Tang, Zhurong et al. (2017). "The material and biological characteristics of osteoinductive calcium phosphate ceramics". In: *Regenerative biomaterials* 5.1, pp. 43–59.

Vacanti, C.A. (2006). *The history of tissue engineering*. DOI: 10.1111/j.1582-4934.2006.tb00421.x.

Yamaguchi, T. (2000). *Clinical Application of Computational Mechanics to the Cardiovascular System*. Springer Japan. ISBN: 9784431702887. URL: https://books.google.es/books?id=p5CpZogEKHgC.

Yang, Z.C. (2019). *Finite Element Analysis for Biomedical Engineering Applications*. CRC Press. ISBN: 9780429592157. URL: https://books.google.es/books?id=7tSNDwAAQBAJ.

Yourek, Gregory et al. (2010). "Shear stress induces osteogenic differentiation of human mesenchymal stem cells". In: *Regenerative medicine* 5.5, pp. 713–724.